68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30:-SEK

$2.95 USA

* Motorola 68020 *
6809-68008-68000-68010

VOLUME VIII ISSUE VII • Devoted to the 68XX User • July 1986

"Small Computers Doing Big Things"

PHOTO CREDIT: NASA

0 74470 12810 07

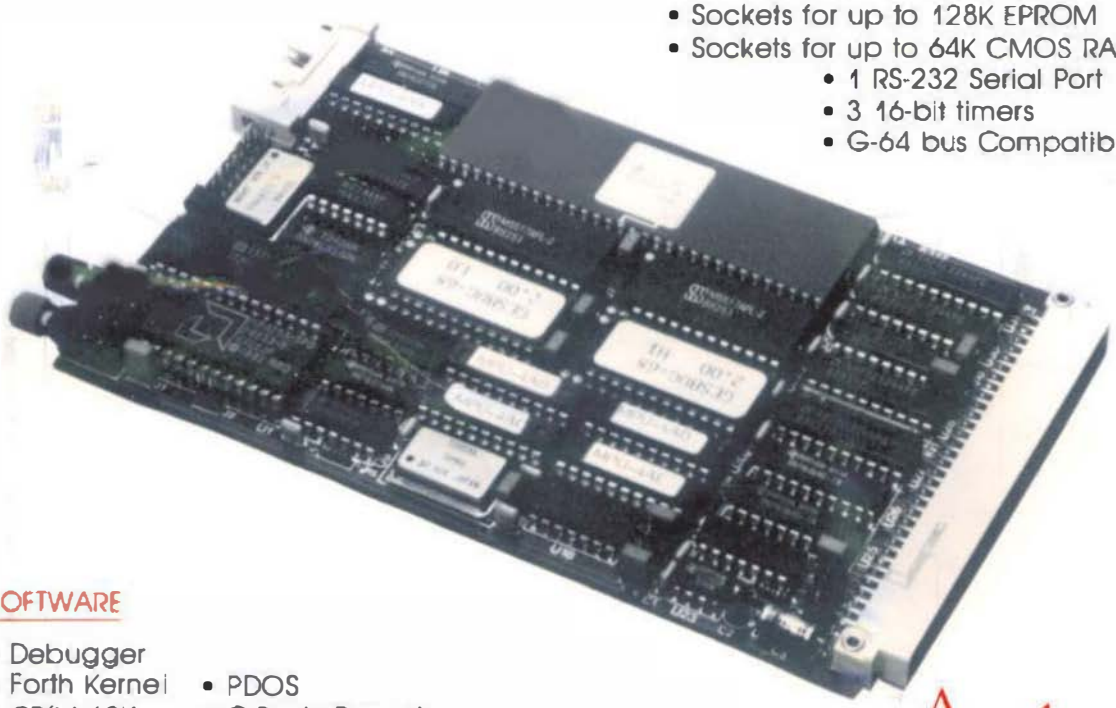
68000 CPU FOR $395

QUANTITY ONE

GESMPU-4A

FEATURES

- 8 MHZ, 16/32-bit 68000 CPU
- Sockets for up to 128K EPROM
- Sockets for up to 64K CMOS RAM
- 1 RS-232 Serial Port
- 3 16-bit timers
- G-64 bus Compatible



SOFTWARE

- Debugger
- Forth Kernel
- CP/M 68K
- OS-9
- PDOS
- C-Basic-Pascal
- Editor-Assembler

OS-9 AVAILABLE

Call for your free data sheet and
our 100 page catalog of board level products.
1-800-443-7722

Gespac

USA — CANADA
100 West Hoover Ave.-11
Mesa, AZ 85202
Tel. (602) 962-5559
Telex 386 575

INTERNATIONAL
3,chemin des Aulx
CH-1228 Geneva
Tel. (022) 713 400
Telex 429 989

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 15 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply $1698.88
60MB Streamer (UniFLEX-020 only) $2400.00
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOS w/battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/battery $998.72
#3116 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS

GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included			
FLEX	Included	Included	Included			
GMXBUG Monitor	Included	Included	Included			
Basic 09, RunB (OS9)	Included	Included	Included			
RMS (OS9)	Included	Included	Included			
DO (OS9)	Included	Included	Included			
VDisk for FLEX	N/A	Included	Included			
RAMDisk for OS9	N/A	$125 option	Included			
O-FLEX	N/A	$250 option	Included			
Support ROM	N/A	N/A	Included			
Hardware CRC	N/A	N/A	Included			

Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX inc.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Soft-ware:

COMPUTERS – HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S-9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Contents

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add $9.50 a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fl** fill and .nf no fill. Also please **do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP Installed Systems World-Wide — OVER 10 YEARS OF DEDICATED QUALITY!

CPI A Division of Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

MUSTANG-020-

FEATURES

- 12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path processor
- 32-bit wide data and address buses, non-multiplexed
- on chip instruction cache
- object code compatible with all 68XXX family processors
- enhanced instruction set - math co-processor interface
- 68881 math hi-speed floating point co-processor (optional)
  direct extention of full 68020 instruction set
  full support IEEE P754, draft 10.0
  transcendental and other scientific math functions
- 2 Megabyte of SIP RAM (512 x 32 bit organization)
- up to 256K bytes of EPROM (64 x 32 bits)
- 4 Asynchronous serial I/O ports standard
  optional to 20 serial ports
  standard RS-232 interface
  optional network interface
- buffered 8 bit parallel port (1/2 MC68230)
  Centronics type pinout
- expansion connector for additional I/O devices
  16 bit data path
  256 byte address space
  2 interrupt inputs
  clock and control signals
  Motorola I/O Channel Modules
- time of day clock/calendar w/battery backup
- controller for 2, 5 1/4" floppy disk drives
  single or double side, single or double density
  35 to 80 track selectable (48-96 TPI)
- SASI interface
- programmable periodic interrupt generator
  interrupt rate from micro-seconds to seconds
  highly accurate time base (5 PPM)
- 5 bit sense switch, readable by the CPU
- hardware single-step capability
- mounts directly to a standard 5 1/4" disk drive

Size 8 15/16 x 5 7/8

These *hi-speed* 68020 systems are presently working at NASA, Atomic Energy Commission, other Government Agencies as well as Universities, Business, Labs, and critical applications centers, Worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a *must*!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information.

```
          MUSTANG-020 Benchmarks **
               Time  Seconds

     Type System                    32 bit      Register
                                    Int. Loop   Long Loop

IBM AT  7300 Xenix Sys 3              9.7        No Registers
AT&T 7300 UNIX PC 68010               7.2        4.3
DEC VAX 11/780 UNIX Berkley 4.2       3.6        3.2
DEC VAX 11/750   "    "    "         5.1        3.2
68000 OS9 68K 8 Mhz                  18.0        9.0
68000  "   "  10 Mhz                  6.5        4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz  2.2        0.88
MUSTANG-020 68020 MC68881 UNIFLEX "   1.8        1.22

** Loop:  Main()
          {
            register long i;
            for (i=0; i < 999999; ++i);
          }

          Estimated MIPS - MUSTANG-020 - 2.5 MIPS
        Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz
```

MUSTANG-020™ Software

OS-9

OS-9	$350.00
Basic09	300.00
C Compiler	400.00
Fortran 77	400.00
Microware Pascal	400.00
Oregonsoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph,Spell,Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Son-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00

Standard MUSTANG-020™ 12.5 Mhz	
Add for 16.6 Mhz 68020	400.00
Add for 16.6 Mhz 68881	400.00
8 Port expansion RS-232 (total of 20 serial ports supported)	498.00
Expansion for Motorola I/O Channel Modules	$195.00

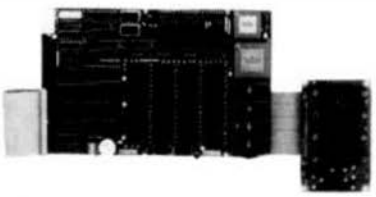
Sculptor+: We are USA distributors for Sculptor+. Call or write for site or multiple system license & discounts, OEM/Dealer.

Special for complete system buyers Sculptor+ - $695.00, Save $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

MUSTANG-020 System component prices - Effective March 1986
Prices subject to change - call for latest quotes.



MUSTANG-020 SBC	$2750.00
Cabinet	$299.95
5"-80 track floppy DS/DD	$269.95
Floppy cable	$39.95
OS-9 68K	$350.00
Winchester cable	$39.95
Winchester Drive 25 Mbyte	$895.00
Xebec H/D controller	$395.00
Shipping USA UPS	$20.00
Total:	$5059.80
Complete Sys order 5% Discount	$252.99
Complete Sys Price Less	$4806.81

This price subject to increase Summer '86

ADD: ->

UniFLEX	$100.00
MC68881 f/p math processor	$495.00
16.67 Mhz MC68020	$400.00
16.67 Mhz MC68881	$400.00

Note: Current OS-9 does not handle the MC68881 - Future revisions will. If the 68881 is anticipated in the future, it must be ordered with the system, when originaly ordered. UniFLEX does support both the enhanced code of the 68020 and 68881 now.

DATA-COMP
10 YEARS OF DEDICATED QUALITY!

PAT - JUST

PAT

With 'C' Source

$229.00

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included**. Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

FLEX
User Notes ?

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi. 48105

I might well be changing the title of this in the near future. It all started with that recent purchase of the Mustang-020 computer. While I want to make it clear that I have no plans to abandon my familiar old FLEX system and all the nice things that it can do, and that I can do with it, I am VERY much impressed with the Mustang, the GMX board that is the heart of it, the Microware OS-9 operating system, and the Microware C compiler.

This is a first for me. I am editing this column using Stylo (Don Williams has sent it to me for review) on the Mustang system. I am highly prejudiced toward my own editor effort PAT, of course, but I have to say that Stylo is my second choice. There have been several improvements in Stylo over the course of its existence, but more about that in a review coming soon.

(Editor's Note: Gotcha Ron, this is our first issue using Stylo to drive a laserwriter. Sure makes your column look prettier. Roger, Carl and Stanley at Great Plains are getting things together, and in a big way. We took your file, dumped it to a Mac by wire from a S50 system, made 4 or 5 command changes to your original Stylo file, sent it to the laserwriter and about 2 minutes later we had your entire article, just as you see it now! Some act, huh?

Actually we have one of the largest and most modern typesetting plants in the Southeast, but, in the beginning (and I do mean beginning, as we are now the third oldest computer magazine in the WHOLE WIDE WORLD!) we promised to do it with our computers. After all, that is what 68 MICRO JOURNAL has always been about.....WHAT CAN BE DONE WITH 'OUR' COMPUTERS! This thing is all 68XXX, and we kept the faith. And I might add that the inputting and editing, because of Stylo, is a lot easier than our 100 grand plus systems. Thanks Ron for letting me bust in, but I just could not resist the temptation. - DMW Now.....

Back to the subject of the Mustang-020 and some discussion of its impressive performance. I've extended my compile time and object code tests for JUST, so I will give you all the information again. JUST is my text processor written originally in Pascal, rewritten and expanded in PL/9 and later translated to "C". First, for the 6809 system, I have a 2 mhz processor and 8 inch floppy drives. PL9 compiled JUST (including the loading of the compiler and the source file to memory) in 35 seconds. The object code is 4400 bytes. McCosh C on the same 6809 system compiled the C version of the same program in 5 minutes and 5 seconds, with an object code size of 8300 bytes. I became interested and set the system at work to run at 2 MHz with the Cal Comp hard disk system. It compiled the JUST.C program with McCosh C in 1 minute 54 seconds. My efforts to use RAMdisk on that system to speed things up further, didn't. First of all, I couldn't run the Hard Disk and the Ramdisk at the same time. Secondly I have only 256K of extended memory so I couldn't put all the C compiler parts and the JUST.C source on the ramdisk and have room for the intermediate files that the C compiler generates. My best results were with the c compiler on the floppy but using the ramdisk for work space for the intermediate files. With the system running at 1 MHz the compile time was 3 minutes 32 seconds. The Tandy-1200, a true IBM 8086 based clone, took 4 minutes 23 seconds to compile JUST using Lattice C. The new information is that the Tandy 3000 using the 80286 compiled JUST with Lattice C in 1 minute 15 seconds. I should mention that the same version of Lattice C was used on both Tandy machines. It is probable that an 80286 optimized version would run considerably faster. I should also mention that the lattice C version generated 16K of object code. Now for the Mustang. It compiled JUST in Microware C in 19.5 seconds! Only 11K of code was generated. I'd say the 68020 looks to be far superior to the other folks advanced processors.

Along with better capabilities and more computing power, with OS-9 you get true multi-user capabilities and the capability to do such things as print a 50 page listing while editing another file. I was going to mention the possibility of compiling a program while doing something else also, but the compile is so fast that you really have to HURRY to do anything else of significance while running a compile as a background task. For example, the C version of PAT for the 68020 compiles in just 1 minute and 15 seconds. The source listing is now about 43 pages, and the object code about 23K. PATC for OS-9/68K is just about finished, and it has a 100K edit buffer. All 43 pages of its source file can be edited in one large file of about 45K of text. The C compiler handles compiling it all in one chunk also. I've done a few little short utilities in C such as one to turn off the glaring status line on a Televideo terminal. It is about 16 lines of putchar() statements to configure the terminal on startup. Such a program compiles in just a few seconds!

Some time ago, I said in a column that I had tried OS-9 on a Color Computer, and that is was about like putting a 1000 horsepower engine on a row boat. Here, though, is the perfect combination. The 68020 and OS-9 complement each other nicely. I have not begun to explore all the things I can do with the combination. There will be more here from time to time as I get more familiar with the system. It has been here for just over a month now, and I have used it considerably in that time.

FLEX

I had to get on to some FLEX topics so as not to leave any of you feeling deserted. I received a letter from a reader in Sweden, Roland Stellrot, who tells me that his computer involvement is as a hobbyist. He had a number of questions regarding FLEX and some of the associated software. There was enough in his first letter to provide me with material for a number of columns, and I answered several questions that didn't require long involved answers in a return letter. I also

mentioned to Roland that my grandfather was born in Goteburg, the city where he works. Roland has sent me information about Goteburg and indicated that he lives in a suburb.

One of Roland's questions concerned the operation of the assembler. In particular he was puzzled over the assembler directives. He said that he could in general understand the instruction set of the 6809, but the assembler directives were hard to figure out. My friend Albert McDaniel (another contributor to '68' Micro Journal) called a few weeks ago with a related question. Both were puzzled over the FCB, FDB, and RMB directives in the assembler, but for slightly different reasons. First of all, I would like to clear up one point of great confusion. The authors of FLEX unfortunately chose to call one of the main parts of their disk interface scheme a File Control Block. The name is unfortunate because of the obvious Mnemonic FCB. Those initials in the context of a discussion of a FLEX file, always refer to the File Control Block, a block of memory 320 bytes long, that is used by the file management system. The FMS uses one file control block for each disk file that is open.

The Assembler has a mnemonic called FCB, that means something entirely different. In this case the FCB stands for "Form Constant Byte". The assembler has a similar instruction FDB or "Form Double Byte". These assembler directives simply are used to place a specified code or bit pattern at the current program counter location in memory. That sounds like a mouthful, so let me give an example. Please remember that I am looking for a simple example and that there are other ways to accomplish the same results. Suppose you need a quick way to convert one sort of code into another. For a dumb example, take a binary number from 0 to 9 and convert it into the ASCII code for that number. One way is simply to add $30 to the number, but there are other code schemes that don't have such a nice correspondence. Ignoring the fact that there is a simple way, we could use a table. Listing 1 shows a subroutine that is passed the binary value in the B accumulator and returns the ASCII character in the A accumulator.

```
CONVRT LDX #TABLE
    LDA B,X
    RTS

TABLE  FCB $30,$31,$32,$33,$34,$35,$36,$37,$38,$39
```

The scheme here ought to be clear. Point the X register at the TABLE and load the A accumulator with the B'th item in the table. if B is zero, A will be loaded with $30, the code for zero. If B is $09 A will be loaded with the value at 9,X or $39, the ASCII code for printing the digit 9. The assembler is a little smart, and the above code doesn't take advantage of its capabilities. Table could just as well be defined as below:

```
TABLE FCB '1,'2,'3,'4,'5,'6,'7,'8,'9
```

The assembler accepts a character preceded by an apostrophe and recognizes it as meaning the ASCII value that represents that character. By now you are wondering about the FDB instruction. You could conceivably use it for TABLE, but its use would obscure what you are trying to do rather than clarify it.

```
TABLE FDB $3031,$3233,$3435,$3637,$3839
```

Though the result would be the same, the intent is lost. What would you use double bytes for? They represent 16 bit values such as addresses. I won't go into the use of a JUMP table here, but you would use FDB in constructing a table of addresses for, say, entry points into a program module, perhaps at the beginning of the module. You don't have to use literal hexadecimal values as I did above. If your program contains labels, the "value" of a label is its address. That is, FDB CONVRT, will generate a 16 bit value that is equal to the address in your program of the label CONVRT. If that is not clear, let me know and I'll try again.

While we are looking at "constants", since that is really what the tables above are all about, there is another kind of constant called a string constant. Anyone familiar with BASIC should have an idea what a string is. It is a "string" of characters.

```
MESSAG FCC /This is a string/
    FCB $04
```

That will generate the ascii characters in sequence to spell out just what you see there. Most 6809 FLEX programs use the convention that the end of a string is signaled by using the Hex value 04. That is simply because there is a FLEX routine called PSTRNG (see a recent column) that will print that line of text and stop when it finds a $04. LDX #MESSAG followed by JSR PSTRNG will get you a CR (PSTRNG issues a CR before printing a string) followed by the message. I don't want to complicate things nor do I want to confuse you unduly, but this brings up another way to generate our TABLE of the first example above:

```
TABLE FCC /0123456789/
```

Again, all the methods shown here are equivalent. The reason for using any one of them would be clarity for yourself or someone else reading the program. You could classify all the assembler directives we have discussed so far as useful for generating CONSTANTs in your program. BASIC has no equivalent of a CONSTANT unless it might be the assigning of a value to a variable and leaving it alone throughout the execution of a program. Pascal or C have things called Constants. In Pascal you use the keyword CONST and then list them. CR = CHR(13); etc. In C you define them as in #define CR 0x0d. (0x0d in C is equivalent to $0D in the assembler).

You probably are all familiar with how in BASIC you define an array with a DIM statement. DIM A(10) would be an example. Suppose you want a working space in an assembler program to store ten 16 bit values in an array. You don't know what values are to be stored there. The program will change them as it progresses, and you don't care what is there initially because you are going to store values in the array or table in the process of running your program.

```
TABLERMB 20
```

That statement will Reserve Memory Bytes for you. The argument is 20 indicating to the assembler to save space for 20 bytes or 10 16 bit values. The Label TABLE will have the value of the address of the table. LDX #TABLE will point X at the first value in the table. If B contains the "index" of the array element you need to access (the first one is index 0), you simply ASLB to multiply it by 2 and LDD B,X. The multiply by 2 is because each entry is two bytes long. The item at index 4 is located at 8,X. You might wonder what

RMB does, if it generates no code whatever. If TABLE happens to be in the middle of your program somewhere, it simply advances the assembler's program counter by 20 so that these 20 bytes are not used for program code, which brings me to the next item for discussion.

The other question that was asked me was more or less "How can you just put a table of data in the middle of the program?" In other words, "How does the program know not to try to execute the bytes in the table?" The answer is simple but not obvious. You can just stick a table anywhere in a program if you are careful that the last instruction before the table is a BRA or JMP to a label just past the table. It doesn't make much sense, however, just to stick a table in the middle of a section of code and jump around it. Rather, a table is usually put before any executable code, after all executable code, or it is tucked in between subroutines. A very good place to put a table is just before the subroutine that uses it. In other words, keep it close to the code with which it is associated. You will always see a JMP, BRA, or RTS instruction just before a table if it appears to be in the middle of a section of code. Further, the next line after the table will have a LABEL. If it did not, it could never be executed since there would be no way to JMP or JSR to it!

This discussion is not quite complete without mention of the other assembler directive used to define numbers for the assembler. That is the EQU directive. Actually, the EQU more closely parallels the #define of C than the FCB or FDB do. Suppose I am writing a program that needs to return to FLEX after it is finished. The last statement would be a jump to FLEX WARMStart location which is $CD03. Though JMP $CD03 would work perfectly well, it is clearer to define WARMS at the start of the program as WARMS EQU $CD03, and at the end of the program use JMP WARMS. There is another reason besides that of clarity for using such a scheme. Back in the "olden days" of Miniflex and the 6800, WARMS was defined as $AD03 for Miniflex. A lot of users wrote their own little utility programs for Miniflex, and if they were smart and used names for the FLEX entry points, all they had to do was change a bunch of $A to $C once for each entry point definition using the EQU directive, and their program would assemble and run under FLEX09! If the program had $ADxx addresses strung throughout, the edit for the change was much harder. This discussion is the same one that comes up with BASIC. If you give a program constant a name, it is easy to change the program parameters by a simple edit rather than a long complicated "hunt" for all the places where you used that value or number.

I hope this discussion has been of some value to you. I am going to send a "preview" copy of this to Roland Stellrot in appreciation for his bringing up the question.

Dvorak Keyboard

The February issue of Byte had an article on the Dvorak keyboard. Dvorak was a professor of statistics at the University of Washington. He realized that the typewriter keyboard had been purposely made inconvenient because early typewriters had a tendency for the keys to jam. The keyboard was set up with the most commonly used letter pairs purposely separated so the typist would have to slow down. Dvorak put the most commonly used letters in the "home row" with all the vowels at the left and the most used consonants at the right. I had heard about the Dvorak keyboard for years, and always had wondered about it. A couple of days ago, reading the article in BYTE prompted me to write a C program to convert the keyboard. It sees the keys that you type, but it echoes what would be there on a Dvorak. The key layout is in the Byte article or you can look it up in the library, (or for that matter, hit keys and make yourself a map).

I should point out that the code conversion table used in the DVORAK program would be a good example of something more useful to do with a table in assembler in the discussion above.

I've given it a pretty good go, having memorized the positions of all the letters. The top row of numbers and punctuation is exactly the same, so there is little to learn there. At any rate, I am able to type on the Dvorak arrangement by "touch" so I didn't have to rearrange the key caps on my terminal. I can breeze along at about 7 or 8 words per minute presently. I think a little practice every day will get me familiar with it so I will be able to switch between qwerty and Dvorak just as one can drive two different cars without a great deal of thought after some mental shifting of environments. After doing the C program, I did a new version of PAT with the conversion built in to the get character routine. I spend half an hour typing a couple of paragraphs that would take me a few minutes with the qwerty arrangement.

The C program is included here so you can type it in and try it. It should run in just about any C including the TINY varieties. I would be interested in any comments from any touch typists who try the Dvorak keyboard. I'll practice now and then (a long session gets too frustrating) and keep you posted as to whether I think I would ever get faster using it than the qwerty keyboard that I have been using since my high school typing course.

RA

```
/* echo characters to terminal for setting up terminal */

#include <stdio.h>

/* qwerty keys in comments above Dvorak keys */
char chartab[] =

/*  sp  !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /  */
  ' ','!','_','#','$','%','&','-','(',')','*','+','w','[','v','z',

/*  0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?  */
  '0','1','2','3','4','5','6','7','8','9','S','s','W',']','V','Z',

/*  @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O  */
  '@','A','I','J','E','>','U','I','D','C','H','T','M','M','B','R',

/*  P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _  */
  'L','"','P','O','Y','G','K','<','O','F',':','/','\\','=','^','{',

/*  `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o  */
  '~','a','x','j','e','.','u','I','d','c','h','t','n','m','b','r',

/*  p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~  del */
  'l','"','p','o','y','g','k',',','q','f',';','?','+',']','^',0x7f };

main()
{  char ch,chi;
   stdout->_flag |= _UNBUF;
   stdin->_flag |= _UNBUF;
   system("tmode novcho nopause abort=0 quit=0 xof=0 xon=0 xoff=0");
   do
   {  ch = getchar();
      if (ch==0x0d) (putchar(chi) )
      if(ch == 0x80) (putchar(ch); putchar(0x20); putchar(chi); )
      if (ch)=0x20)
      {  chi = ch -0x20;
         chi = chartab[chi];
         putchar(chi);
      }
   } while (ch != 0);
   system("tmode xoreal");
}
```

Basically OS-9 ™

By: Ron Voigts
2024 Baldwin Court
Glendale, Il 60139

Working With The System

Many years back, before I was running OS-9, I found myself with a programming dilemma. I was working with a system on a S-100 bus, running a Z80 micro-P. I had written a Basic program that involved some I/O over a home brew bus to a piece of test equipment. Alas, the test equipment proved to be too fast! Or should I say the Basic was too slow! I proceeded to write the input/output routines in assembly code. My intention was to keep the basic program for the higher level intervention and use the machine code I had generated for talking to the equipment. Sounds simple? Well it wasn't!

The inventors of the system had never planned such a thing. You could run Basic. Or you could run machine code. There wasn't any compromise, as far as the system was concerned. I immediately sought out a computer guru. A man whose knowledge of such things was impeccable. This was someone who had come to grips with the problem and survived. His advice was to "work around the system." He had solved the problem by writing a routine to load the machine code, the Basic interpreter, the Basic program, adjust the pointers and execute the whole mess. I being one to shun excessive ambition went the cowards way out. I loaded my program from Basic by poking it into memory. I also had to poke in a few pointers so Basic wouldn't overwrite it and I was all set. I had learned how to "work around the system".

Good news! I haven't had to work around the OS-9 system. With OS-9 you work with it. Basic09 is available, but it doesn't restrict what you can do. Most times you can find a way to do what you want with Basic09 or what language you like to work in. If not there is always writing your own routines and using them. You may want to do something faster than what is available in Basic09. Although I have found Basic09 to be very fast. And you might have some special need that just wasn't thought of when Basic09 was created. Whatever you want to do, you can work with the system. The languages, the commands, the system calls and everything works together.

It is very seldom that I find something I can't do with Basic09. The language is well written. It handles everything a good Basic language should and much more. Occasionally something will come up that you cannot do or it may be something that you want to do a little differently. A good example is DATE$. It is a psuedo string that contains the date and time. From a program you would treat it as a string. A line in a program like:

```
PRINT DATE$
```

might return

```
86/15/12  12:24:38
```

If you wanted to use the individual parts of the date and time you would have to extract them. Not necessarily a difficult task, but a little messy. You would need to used lines such as:

```
year:=VAL(LEFT$(DATE$,2))
hour:=VAL(MID$(DATE$,10,2))
```

I think everyone has used lines like these, but wouldn't it be nice to read them directly. Listing 1 has a short subroutine that will return the date and time into a complex data structure. The set up for it in a program would appear:

```
T Y P E                 t i m e                 =
year,month,day,hour,minute,second:BYTE
DIM t:time
```

Using it, you would enter:

```
RUN getime(t)
```

and everything would be returned in the 6 byte time packet of t. So, the year would be t.year, the month, t.month and so forth.

This subroutine is useful for showing how the parameters are passed to a Basic09 procedure. Everything is passed on the stack. If you're not familiar with stack operations let me tell you a little about them. If you are familiar with them, feel free to plod ahead

The stack is a reserved area in memory that can be used for last-in/first-out storage. A special register, called the stack pointer or SP for short, points to the last entry. Pushing a byte on the stack involves decrementing the SP register and storing the byte at the new location it points to. Pulling a byte off the stack reverses the procedure. If a number of bytes are stored, when they are removed they come off in the reverse order they were put on. The stack has a number of excellent uses. When entering a returnable subroutine, the program counter is pushed on the stack. Then execution can be directed to the subroutine. When finished PC is pulled back off the stack and the program begins execution where it was before the call. Registers can be temporarily stored on the stack. The stack can be used a place of variable storage without having to provide specific memory in the data area. Data can be referenced by its offset from SP.

A Basic09 subroutine uses some of these. Parameters are passed by pushing variable location and size on the stack. The parameter count is pushed on and a call is made to the subroutine. The subroutine in Listing 1 is easy to follow. There is only one parameter to pass. The size of the parameter is first pushed on the stack. It is a two byte integer value. In our case, it is $0006, since there are 6 bytes in the time packet. Next the address of the time packets location is pushed, again it is a two byte value. Third, the number of passed parameters are pushed in two byte form; here, it is $0001. Finally, the call is made to Getime and the return address is pushed on it.

This makes using them in Getime a simple task. Return is at 0 bytes from SP, Count is at 2 bytes from SP, Time is at 4 bytes and Size is at 6 bytes. So, 'ldd Count,S' means load register D with the value at location SP+2. The other ones can be evaluated similarly. Notice that the line 'Done' uses 'rts' to return to the caller. This pulls the last two bytes off the stack and puts them back into the PC register. The program continues execution from where it started. We could have also used a 'puls pc', which would have had the same effect.

If more than one parameter is in the parameter list, these parameters are pushed on the stack in reverse order in which they appear. Let us say you pass three integers--A,

B, and C. The size of C would first be pushed, and then it's location. Next, B's size would go on the stack and it's location. Finally, A's size and location would be pushed. $0003 would be pushed to indicate that there are three parameters passed and then the call would be made. This system is very flexible. You can pass a variable number of parameters. You can mix integers and byte values, since the size of them is known. As long as the subroutine you write is aware of what is being passed, you can do almost anything.

Speed can be another consideration. I did some benchmark's quite awhile ago and found that Basic09 was indeed fast. Usually it ran 4 to 10 times faster than Basic found on the Color Computer. I am sure I don't have to convince anyone already running Basic09. But there comes a time that you really need more speed. I wrote a simple bubble sort to sort 100 integers in ascending order. It used 2 FOR...NEXT loops. And swapped values as it went along. The time it took sorting the 100 numbers was about 16 seconds. Surely, we can do it faster! Listing 2 show a faster way.

This sort routine is written in C code. C also passes parameters on the stack in 2 byte chunks. This time we identify the parameters in the argument list following 'sort'. They are 'count', 'array', and 'asize'. We don't identify a return since C knows it's there. Notice the declaration:

```
int *array, asize;
```

We declare 'array' to be a pointer to some area holding integers and asize tells how many are there. 'Count' does not even get a mention, since it will not be used. Its sole purpose was to be a position marker in the argument list. This little subroutine is almost identical to the Basic09 version I wrote with a few exceptions. The extra variables are of the type 'auto', so they also get stack space. And one variable is of the type 'register' to help improve speed of execution. Another difference is that this routine can sort any size array. The Basic09 version would be limited by what was declared in the PARAM statement. The execution time for this bubble sort with 100 integers was about 4 seconds. Considerably faster than the Basic09 version.

By the way, this little subroutine written in C code must be compiled in a particular way for use by a Basic09 program. My original source was named 'sort.c'. So first, I compiled it to a relocatable module with:

```
cc1 sort.c -rs
```

The '-rs' option told the compiler not to link the module, leaving it in relocatable object format and not to generate stack checking code. Next I linked it by entering:

```
c.link sort.r -b=sort -o=sort
```

The '-o' tells the name of the output file. The '-b' tells the name of the function that Basic09 will use when it executes a RUN.

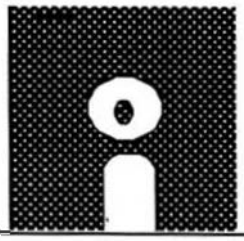
I think you'll find Basic09 covers most programming needs. But when you need to create code outside of it, you'll find it's easy to do. Perhaps you'll need to do something out of the ordinary or just a little different. Maybe, you'll be trying to improve speed of execution. Whatever you do, isn't it nice to know the system works with you? And you don't have to work around it?
RV

```
           **************************************
           *
           * Getime,  a subroutine for getting
           * time in a BASIC09 program
           * by Ron Voigts,  15-MAR-86
           *
           * To Use:
           *     RUN Getime(t)
           *     and
           *     DIM time=year,month,day,hour,minute,second:BYTE
           *     DIM t:time
           *
           *
           *
                                 nam   GetTime
                                 ttl   Returns the time from Basic09
           *
  0021                 Type      set   Sbrtn+Object
  0081                 Revs      set   Reent+1
           *
  0000 87CD0033                  mod   TimeEnd,TimeName,Type,Revs,Start,0
           *
           *
           * Use /r/defs/Os9Defs
                                 ifp1
                                 endc
           *
           *
           * The module's name
  000D 47655469        TimeName  fcs   /GeTime/
  0013 01                        fcb   $1          Edition number
           *
           *
D 0000                           org   0
D 0000                 Return    rmb   2           Return adress
D 0002                 Count     rmb   2           Count of Parameters
D 0004                 Time      rmb   2           New Directory
D 0006                 Size      rmb   2           Size of name
           *
           *
           * Start the module here
  0014                 Start     equ   *
           *
           * Check for correct number of parameters
  0014 EC62            Pcheck    ldd   Count,S     Get parameter count
  0016 10830001                  cmpd  #1          Were there 1?
  001A 260E                      bne   Perror      No, then an error
  001C EC66                      ldd   Size,S      Get Packet size
  001E 10830006                  cmpd  #6          Are there 6 bytes?
  0022 2606                      bne   Perror      No, then an error
           *
           * Change the time
  0024 AE64            Change    ldx   Time,S      Pointer to time packet
  0026 103F15                    os9   F$Time      Set the time
           *
           * If carry is set we'll go back with an error!
  0029 39              Done      rts               Return to Basic09
           *
           *
           * Return a parameter error
  002A C638            Perror    ldb   #56         Get parameter error
  002C 1A01                      orcc  #%00000001
  002E 20F9                      bra   Done        return with an error
           *
           *
  0030 529318                    emod
  0033                 TimeEnd   equ   *
                                 end

00000 error(s)
00000 warning(s)
$0033 00051 program bytes generated
$0008 00008 data bytes allocated
$1E8E 07822 bytes used for symbols
```

```
.ce
*LeIeBeTeIeNeGe #2

/* Subroutine to sort integers */
sort(count, array, asize)
int *array, asize;
{
    register int i;
    int j, k;
    for (i=0; i<asize; i++)
        for (j=i; j<asize; j++)
            if (array[i]>array[j])
            {
                k=array[j];
                array[j]=array[i];
                array[i]=k;
            }
}
```

C User Notes™

By: E. M. (Bud) Pass Ph.D
Computer Systems Consultants
1454 Latta Lane, N.W.
Conyers, Ga 30207
404 483-1717/4570



INTRODUCTION

This chapter continues the discussion of C compilers and the description of the construction of an editor.

C COMPILER INFORMATION

Microware reports current known bugs in a Vendor Bug Report List. None of the other 6809 or 68000 vendors make such a list available. I praise Microware for this action, and hope the others will follow. Some of the more important problems are listed below.

The following problems with the Microware os9/6809 C compilers were reported in Microware's Vendor Bug Report.

> The "signal()" function does not work properly. Microware suggests using "intercept()", as "signal()" will not be supported on future releases. The UNIX V and new ANSI C standards both call for the use of "signal()", not "intercept()", so Microware's comment is surprising.
>
> The profiler does not work due to the absense of the module "eprof". Microware will supply a patch upon request.

The following problems with the Microware os9/68000 C compiler were reported in Microware's Vendor Bug Report.

> Casting to a smaller data type in many instances does not produce the correct code. Microware stated that this was a known bug and is fixed in version 2.0 of the C compiler. See the example program at the end of this chapter.
>
> There are several problems with the "-dname" compiler option. One of these silently causes incorect code, the other gives an error message as well as correct code. Microware suggests to avoid this compiler option temporarily. The #define statement works correctly. This was also reported by Bob Larson.
>
> Structures containing strings actually contain pointers to strings, not the strings themselves. The "sizeof()" function provides the correct element size, given this condition. However, the user must be careful in moving structure elements within memory or to of from I/O devices, as the data in the strings is not present within the structure elements themselves. This same situation is true in all known current versions of the McCosh C compiler.
>
> A "case 0:" in a switch statement sometimes generates incorrect code. The person reporting the problem suggests prefixing the "case 0:" statement with a dummy case statement.
>
> The C compiler generates incorrect code if a "return" statement is executed as part of an "if" statement inside of a "for" loop.
>
> The "register" declaration sometimes causes the generation of incorrect code.

The following problem with Microware os9/68000 C compiler version 1.3 was reported by Bob Larson, although not by Microware.

> The meaning of the value returned by the "getstat" function to determine the number of characters waiting is different between the Microware 6809 and 68000 C compilers. Zero means "at least one character is ready" on the 6809, and "no characters are ready" on the 68000. This difference was found only by comparing the two manuals.

One problem with the Microware C library which is documented in the os9/68000 C compiler manual involves a buffered file opened for update ("r+" or "w+"). The program is required to call "fseek()" or "rewind()" when switching from reading to writing or from writing to reading in order to flush the buffer. The UNIX V C compiler requires no such program action, as it automatically flushes the buffer in this case.

THE C USERS' GROUP

I have received several inquiries about the C Users' Group. This is a for-profit (not non-profit) organization which collects, organizes, verifies, and distributes public-domain C code for $8.00 per diskette. The dues are currently $15.00 per year. The address is as follows:

C Users' Group
Box97
McPherson, KS 67460
(316) 241-1065

They are looking for librarians for specific areas. Write to them, not to me, for more information. I have no direct relationship with them, other than being a member of the Group.

LOW-LEVEL CURSES FUNCTIONS

If the curses package is unavailable for a given system, but the terminfo package is available, many of the low-level curses functions may be provided in a reasonably-simple fashion.

The C code below provides most of the low-level curses functions. The following simplifying assumptions have been made:

> The output device has 24 lines and 80 columns.
>
> Terminal attributes are line-oriented, not cursor-oriented.
>
> Any non-normal attributes are displayed as reverse video.

Attribute-change characters occupy zero or one spaces on the screen, but one position is reserved for each attribute change.

Fields (areas between attributes) must start and end on the same line.

Columns 1 and 80 are reserved for attributes.

```
#include <ctype.h>
#include <stdio.h>
#include <termio.h>
#define SGTTY struct termio
#include <term.h>

#define A_ATTRIBUTES   0377600
#define A_BLINK        0002000
#define A_BOLD         0010000
#define A_CHARTEXT     0000177
#define A_DIM          0004000
#define A_INVIS        0020000
#define A_NORMAL       0000000
#define A_PROTECT      0040000
#define A_REVERSE      0001000
#define A_STANDOUT     0000200
#define A_UNDERLINE    0000400

#define X_ATTRIBUTES (A_ATTRIBUTES &
~A_PROTECT)

#define COLS        80
#define LINES       24
#define WINDOW      struct scrstr

#define addch(ch)      waddch(stdscr,ch)
#define addstr(str)    waddstr(stdscr,str)
#define attroff(at)
wattrset(stdscr,winch(stdscr) & ~(at))
#define attron(at)
wattrset(stdscr,winch(stdscr) | (at))
#define attrset(at)    wattrset(stdscr,at)
#define cbreak()
#define clear()        wclear(stdscr)
#define echo()
#define endwin()       resetraw()
#define erase()        wclear(stdscr)
#define flash()        bell()
#define flushinp()     flusher(TCSETAF)
#define getch()        getchar()
#define getstr(str)    gets(str)
#define getyx(win,y,x) (y = win->_cury, x = win-
>_curx)
#define inch()         winch(stdscr)
#define move(y,x)      wmove(stdscr,y,x)
#define mvaddch(y,x,ch)
mvwaddch(stdscr,y,x,ch)
#define mvaddstr(y,x,str)
mvwaddstr(stdscr,y,x,str)
#define mvgetch(y,x)          mvwgetch(stdscr,y,x)
#define mvgetstr(y,x,str)
mvwgetstr(stdscr,y,x,str)
#define mvinch(y,x)           mvwinch(stdscr,y,x)
#define mvwaddch(win,y,x,ch)
(wmove(win,y,x), waddch(win,ch))
#define mvwaddstr(win,y,x,str)
(wmove(win,y,x), waddstr(win,str))
#define mvwgetch(win,y,x)     (wmove(win,y,x),
getchar())
#define mvwgetstr(win,y,x,str)
(wmove(win,y,x), gets(str))
#define mvwinch(win,y,x)      (wmove(win,y,x),
winch(win))
#define nl()
#define nocbreak()
#define noecho()
#define nonl()
#define noraw()
#define raw()
#define refresh()
#define standend()     wattrset(stdscr,0)
#define standout()
wattrset(stdscr,winch(stdscr) | A_STANDOUT)
#define winch(win)     win->_y[win->_cury][win-
>_curx]
#define wrefresh()

struct scrstr
{
    unsigned short int _cury, _curx, _yy, _xx;
    unsigned char _y[LINES][COLS];
};

char *tparm();
int outc();
short int hrow, hcol, hidns, cursor_set = 0;
SGTTY ttycurr, ttyold, ttynew;
WINDOW stdscrm, *stdscr = &stdscrm, *curscr =
&stdscrm;

beep()
{
  put_cap(bell, 0);
}

clearok(win, bf)
WINDOW *win;
short int bf;
{
  short int c, i, j, k, l, x, y;

  if (bf)
  {
    getyx(win, y, x);
    reset_screen();
    for (j = 0; j < LINES; ++j)
    {
      for (i = 0, k = COLS - (!(j < LINES - 1)); i <
k; ++i)
        if (win->_y[j][i] != 0x20)
          break;
      if (i >= k)
        continue;
      if ((j < LINES - 1) && (win->_y[j][i] >=
0x80))
      {
          wmove(win, j, COLS - 1);
          wadjcursor(win, 0);
          outattr(0x80);
          win->_xx = 255;
        }
        wmove(win, j, i);
        wadjcursor(win, 0);
        for ( ; i < k; (win->_curx = ++i))
        {
```

```
            c = win->_y[j][i];
            if (c >= 0x80)
            {
                wadjcursor(win, 0);
                outattr(c);
            }
            else
              if ((c != 0x20) && (isprint(c)))
            {
                wadjcursor(win, 1);
                putchar(c);
            }
            else
              continue;
             win->_xx = i + 1;
          }
        }
         wmove(win, y, x);
      }
}

initscr()
{
    ioctl(0, TCGETA, &ttycurr);
    ttyold = ttycurr;
    setupterm(0, 1, 0);
    ttycurr.c_iflag = IGNBRK | IGNPAR | IXON |
IXOFF;
    ttycurr.c_oflag = ttycurr.c_lflag =
ttycurr.c_cc[VTIME] = 0;
    ttycurr.c_cc[VMIN] = 1;
    ttynew = ttycurr;
    ioctl(0, TCSETAW, &ttycurr);
    cursor_set = strlen(tparm(cursor_address,0,0));
    wclear(stdscr);
}

outattr(at)
short int at;
{
   char *attr;

   put_cap(attr = (at > 0x80) ?
enter_standout_mode :
      exit_standout_mode, 0);
    if ((!attr) || ((int)magic_cookie_glitch <= 0))
      putchar(' ');
}

resetraw()
{
    setterm(&ttyold);
}

reset_screen()
{
    outattr(0x80);
    put_cap(cursor_normal, 0);
    put_cap(clear_screen, 0);
}

setraw()
{
    setterm(&ttynew);
}

setterm(ttywhich)
SGTTY *ttywhich;
{
    ttycurr = *ttywhich;
    ioctl(0, TCSETAW, &ttycurr);
}

waddch(win, ch)
WINDOW *win;
short int ch;
{
    short int x = win->_curx, y = win->_cury;

   if ((x >= COLS - 1) || (y >= LINES))
      return;
    if (!isprint(ch))
      ch = 0x20;
    if (winch(win) != ch)
    {
       wadjcursor(win, 1);
       outc(winch(win) = ch);
       ++(win->_xx);
    }
     ++(win->_curx);
}

waddstr(win, str)
WINDOW *win;
char *str;
{
    while (*str)
      waddch(win, *str++);
}

wadjcursor(win, ca)
WINDOW *win;
short int ca;
{
    short int x = win->_curx, y = win->_cury;
    short int xx = win->_xx, yy = win->_yy;
    short int i = x - xx;

    if ((y != yy) || (x != xx))
   {
      if ((y == yy) && cursor_right && cursor_left
&&
          (i <= cursor_set) && (i >= -cursor_set))
      {
         if (i > 0)
            while (--i >= 0)
               put_cap(cursor_right, 1);
         else
            while (++i <= 0)
               put_cap(cursor_left, 1);
      }

      else
      {
          tputs(tparm(cursor_address, y, x), 1, outc);
      }
       win->_yy = y;
       win->_xx = x;
   }
}

wattrset(win, at)
WINDOW *win;
short int at;
{
    short int x = win->_curx, y = win->_cury;
```

```
  if ((y >= LINES) || (x >= COLS) ||
     ((x == COLS - 1) && (y >= LINES - 1)))
     return;
  if (winch(win) !=
     (at = ((at & X_ATTRIBUTES) ? 0xff :
0x80)))
  {
     if (x == COLS - 1)
       at = 0x80;
     wadjcursor(win, 0);
      outattr(winch(win) = at);
     if (++(win->_xx) >= COLS)
       win->_xx = 255;
  }
  if (win->_curx < COLS - 1)
     ++(win->_curx);
}

wclear(win)
WINDOW *win;
{
  short inti, j;

   win->_cury = win->_curx = win->_yy = win-
>_xx = 0;
  if (win->_y[0][0] != 0x20)
     for (i = 0; i < LINES; ++i)
       for (j = 0; j < COLS; ++j)
          win->_y[i][j] = 0x20;
  reset_screen();
}

wmove(win, y, x)
WINDOW *win;
short int y, x;
{
   if (!((win->_curx = x) + (win->_cury = y)))
  {
     wadjcursor(win, 0);
     ioctl(0, TCSETAW, &ttycurr);
  }
}

outc(ch)
charch;
{
  putchar(ch);
}

put_cap(cap, tc)

          char *cap;
          short int tc;
          {
            char *cp = cap;

            if (((int) cp != -1) && cp && *cp)
            {
               if (tc && strchr(cp, '%'))
              {
                  if (((int) (cp = tparm(cp, 1)) == -1) || (!cp))
                   cp = cap;
              }
               tputs(cp, 1, outc);
            }
          }
```

C PROBLEM

The previous C problem was to find the bug in the b-tree example program related to where "gets(q)" places the string that is read from the standard input device. The following program fragments should help the reader locate the problem:

```
#define STRING 4096 /* Init alloc for strings */
:
#define ADDSTR 2048 /* Addl alloc for strings */
:
  char *q;
:
  q = cstore = alocstr(STRING); /* string space */
  cend = cstore + STRING;
  if (gets(q) == NULL) /* get first line */
    exit(0);
:
  alloc->str = q;
  puts(q);
  q = nextstr(q);      /* get new string pointer. */
  while (gets(q) != NULL)
  {
:
            /* Creating a left subtree */
:
             puts((r->left)->str = q);
            q = nextstr(q);
            break; /* get next line */
:
            /* Creating a right subtree */
:
             puts((r->right)->str = q);
            q = nextstr(q);
            break; /* get next line */
:

/* Returns pointer to 'n' bytes for string storage.
*/
char *alocstr(n)
int n;
{
  char *x;
   if ((x = maloc(n)) == NULL)
  {
      fprintf(stderr, "No memory for strings\n");
     exit(2);
  }
   return (x);
}

/* Returns a pointer to the next available string
space.
  Gets more space if necessary. */
char *nextstr(s)
char *s;
{
  char *x;
   if ((x = s + strlen(s) + 1) >= cend)
     cend = (x = alocstr(ADDSTR)) + ADDSTR;
   return (x);
}
:
```

The bug manifests itself when there are fewer characters remaining in the allocated block of characters than there are

characters in the string being read with "gets(q)". The characters are placed directly into the allocated block, without regard to the number of characters remaining. In almost all cases, the string will overflow the allocated area.

The bug is easier to comprehend in the case of a very short allocation (make STRING 64) and a very long input record (128 bytes). Then the very first "getc(q)" will overflow the allocated area. This type of analysis is quite often useful in such subtle cases.

The problem is very obvious on systems with memory protection, if the address beyond the last-allocated area is outside the program's addressable memory range, as the system terminates the program prematurely.

The problem is substantially more subtle on systems without memory protection. The string overwrites the contents of the memory locations at several addresses immediately greater than the address of the end of the allocated block.

There are at least two good solutions to this problem. Either involves adding a definition of the longest allowable string to be input, in a manner similar to the following:

```
#define MAXSTR 128/* Max string length */
```

and using "fgets(x, MAXSTR, stdin)" rather than "gets(x)", in order to prevent reading a string longer than expected.

The first solution is to allocate a fixed string similar to the following:

```
char x[MAXSTR];
```

into which to place the input characters read by fgets, then call a version of "nextstr(x)" modified as shown below to obtain an address into which to copy the string contents.

```
/* Returns a pointer to the next available string
space.
  Gets more space if necessary. */
char *cnext;
char *nextstr(s)
char *s;
{
  char *x;
  if (((x = cnext) + strlen(s) + 1) >= cend)
    cend = (x = alocstr(ADDSTR)) + ADDSTR;
  cnext = x + strlen(s) + 1;
  return (x);
}
```

Then "strcpy(nextstr(x), x)" will always move into allocated space, assuming that MAXSTR is less than STRING, and cnext is initialized to cend.

This corrects the problem, at the expense of one new fixed string area, some wasted space at the end of each allocated area, and a somewhat slower program due to the time required to move the characters from this fixed area to the allocated block.

The second solution is to increase the size of each allocated block by MAXSTR, to prevent any possibility of placing characters beyond the end of the allocated area, but not to indicate this additional allocation in the pointer to the end of the allocated area. Thus the input characters would be read directly into the allocated area and "nextstr(x)" would be modified as follows:

```
/* Returns a pointer to the next available string
space.
  Gets more space if necessary. */
char *nextstr(s)
char *s;
{
  char *x;
  if ((x = s + strlen(s) + 1) >= cend)
    cend = (x = alocstr(ADDSTR + MAXSTR))
+ ADDSTR;
  return (x);
}
```

This also corrects the problem, at the expense of a somewhat greater amount of wasted space at the end of each allocated area; however, it runs faster than the other solution, since it does not move the input characters twice.

The program may be improved in another manner, normally saving much more space than that wasted by either solution. There is no logical need for separate allocation of the b-tree nodes and the string space, and the two allocation routines may be readily combined into one routine. Not only does this save program code, but it saves space and time, since fewer allocations are made and space is wasted only at the end of each allocation.

The only really tricky part of the combination of these allocations occurs on a computer, such as a MC68000, PDP-11, WE3200, or IBM370, which requires pointers to be aligned on a word or double-word boundary. The simplest means in which to accomplish this is to force alignment to the next address divisible by 4. However, if portablility to such a machine is not a problem, the alignment is unnecessary.

For the next C problem, code a function to print the B-tree after it has been built by the program. Then the program will function both as a sort and as a duplicate-remover. Hint: use both recursion and iteration to simplify and speed up the function.

EXAMPLE C PROGRAM

Following is this month's example C program; it was placed on NewsNet by Bob Larson and illustrates a problem he discovered with several C compilers in the area of casting and double casting. If a reader discovers another C compiler on which this program fails, please send results to Bob Larson or to myself.

```
/*

Program to test sign extention by casting,
 by Robert A. Larson.
   Uucp: ihnp4!sdcrdcf!oberon!blarson

I need to sign extend the eight least significant
bits of an unsigned variable to an int.

Given that char is an 8-bit and signed (not a
completly
portable assumption, but true on many machines),
casting to a char and then to an int will cause the
sign extension I need.

Not wanting an extra temporary variable, I
decided
that double casting would do what I want:
   intvar=(int)(char)unsignedvar;

However, I found that it did not work correctly
on the
Microware os9/68000 C compiler (v1.3) I was
using at the time.
Microware says that their problems are fixed in
version 2.0.
```

```
K&R (section 2.7) and some other C compilers
agree
with my interpretation:
Machine   os   compiler    bad   possible
Vax 750 4.2BSD   cc          0        9
Prime   Primos   cc 19.4    0         9
68000   OS9/68k M.W 1.3    9         9
3B2     UNIX V    cc        0        9
6809    FLEX    McCosh v27 3          3

A C compiler validation suite should include a
program
that tests this use of casts.
This program assumes 8-bit characters, and
character tests
are not valid otherwise.
This program assumes 16-bit shorts, and short
tests are not
valid otherwise.
If sizeof(int) != sizeof(short) and sizeof(int) !=
sizeof(long)
    int to long expansion testing should be added.
*/

/* comment out as appropriate */
/* #define signedchar ** declaration "signed char"
allowed */
#define unsigchar /* declaration "unsigned char"
allowed */
#define unsigshort /* declaration "unsigned short"
allowed */
#define unsiglong /* declaration "unsigned long"
allowed */
        main()
        {
          int bad = 0, possible = 0;
          char c;
        #ifdef signedchar
          signed char sc;
        #endif
        #ifdef unsigchar
          unsigned char uc;
        #endif
          short s;
        #ifdef unsigshort
          unsigned short us;
        #endif
          int i;
          unsigned u;
          long l;
        #ifdef unsiglong
          unsigned long ul;
        #endif

          u = 0x1ff;
          c = (char)u;
          if(c < 0)
            printf("Characters are signed.\n");
          if (((char)u) != c)
          {
            printf("((char)u) != c\n");
            bad++;
          }
          possible++;
          if (((int)(char)u) != (int)c)
          {
            printf("((int)(char)u) != (int)c\n");
            bad++;
          }
          possible++;
          i = (int) c;
          if (((int)(char)u) != i)
          {
            printf("((int)(char)u) != i\n");
            bad++;
          }
          possible++;
        #ifdef signedchar
          sc = (signed char) u;
          if (sc >= 0)
            printf("??? signed char are not signed.\n");
          if (((signed char)u) != sc)
          {
            printf("((signed char)u) != sc\n");
            bad++;
          }
          possible++;
          if (((int)(signed char)u) != (int)sc)
          {
            printf("((int)(signed char)u) != (int)sc\n");
            bad++;
          }
          possible++;
          i = (int) sc;
          if (((int)(signed char)u) != i)
          {
            printf("((int)(signed char)u) != i\n");
            bad++;
          }
          possible++;
        #endif
          if (sizeof(int) > 2)
          {
            u = 0x1ffff;
            s = (short)u;
            if (((short)u) != s)
            {
              printf("((short)u) != s\n");
              bad++;
            }
            possible++;
            if (((int)(short)u) != (int)s)
            {
              printf("((int)(short)u) != (int)s\n");
              bad++;
            }
            possible++;
            i = (int) s;
            if (((int)(short)u) != i)
            {
              printf("((int)(short)u) != i\n");
              bad++;
            }
            possible++;
        #ifdef unsigshort
            us = (unsigned short) u;
            if (((unsigned short)u) != us)
            {
              printf("((unsigned short)u) != us\n");
              bad++;
            }
            possible++;
            if (((int)(unsigned short)u) != (int)us)
            {
```

```
        printf("((int)(unsigned short)u) !=
(int)us\n");
        bad++;
    }
    possible++;
    i = (int) us;
    if (((int)(unsigned short)u) != i)
    {
        printf("((int)(unsigned short)u) != i\n");
        bad++;
    }
        possible++;
#endif
    }
    printf("%d bad out of a possible %d.\n", bad,
possible);
}
```

If anyone has printable example C programs, please send them. You may receive fame, if not fortune.
BP

- - -

XDMS An Example!

Introduction

Recently I had a need to do a complete expense budget for the company that I work for. I wanted the summaries of the budget in formats similar to the "Mainframe" there. I have DYNACALC and I considered using it. It certainly would do the job if I don't exceed the size requirements. I considered using it and finally decided to do the job with XDMS. One of my motivations was to write this article and provide all of you with a glimpse of the power of XDMS! This article only scratches the surface of what can be done with the very powerful and reasonably priced database. The main reason I started this program was to do the budgeting for an employee buy-out of the company I work for. I have since learned that the employee group was not successful in the asset buy-out and the company has been sold. Last Thursday all the employees including myself have been permenantly laid off. Real nice for next weeks Christmas! Looks like I'll have a lot of time to answer any questions you may have about these programs. I hope you can use these programs in your work or modify them as you need. If you don't want want to type them in, send me either a 5 or 8 in disk, a SASE mailer and $15 and I'll copy them over for you.

Jim Gerwitz
7907 E. Wood Dr.
Scottsdale, AZ 85260
(602) 948-9304

Remember-- You have to have your own copy of XDMS version 1.1 to use these programs. I suggest that you buy the latest version as it is absolute DYNAMITE!!

DESCRIPTION OF THE PROBLEM

In my situation, almost all the requirements were predetermined. Departments already had assigned numbers and corresponding names; the numbers were 3 digits and the names did not exceed 35 characters. Account numbers were 4 digits in the Chart of Accounts and were all preassigned. I wanted to be able to request 4 reports. The first one I wanted was "BUDGET BY DEPARTMENT". In this report, I wanted an individual sheet for each department with the department name at the top; all the account numbers in ascending order which had dollars for that department. The headings across the page were to be: ACCT#, NAME, YEARLY (total), PCT (as a percent of the total of all expenses), (amount by month for the months of) JAN, FEB.....NOV, DEC. I also wanted the PCT and YEARLY fields as well as the monthly amounts totaled on each page. The second report that I wanted was a "TOTALS BY ACCOUNT" report. Here I wanted all accounts to be totaled by account number and placed in sequence with a grand total for the fields YEARLY, PCT and JAN, FEB.....NOV, DEC at the end. This report is a summary of another report that we will discuss next, but it is an example that has some benifit. The third report I wanted was "ACCOUNTS BY PARENT DEPARTMENT". This report is one which shows, in account sequence, which department each account came from, and then provides a subtotal for the account and a grand total at the end. You do not need the second report if you prepare this one. I like this report because when you start thru a budget adjusting sequence, you see each account in each department and have an idea where to go to change your data in the database by using the UPDATE command. You can also see the total change in the account subtotals here. The last report I wanted is a little tricky but is needed. It is called SUMMARY BY DEPARTMENT. In reality this is needed to provide a basis for a "pro forma" P & L statement. In the case I have here, it provides a very good example on how to pull various amounts out seperately and summarize totals. I suggest that you review each of the example printouts for format and then read the ".CTL" program discussions carefully to see the things being done. I will discuss each program later in the article.

IMPORTANT

The enclosed programs are written for a Microline 182 printer that can print 132 columns across. If you do not have such a printer, you will have to change the programs to print less columns. I would suggest that you

include only 6 months in a pass, and make two passes on each printing.

HOW TO USE

To use these programs I suggest the following: 1. Using your editor, enter the .CTL files as shown. 2. Using XDMS and the DEFINE command, define the 3 .DMS files, named BASE, ACCTS and OPSDEP. 3. Place these .CTL and .DMS files on a freshly prepared bootable system disk, along with a minimum number of .CMD files. Have the STARTUP file define the Work and System drives as 0 (zero). 4. Per the XDMS manual, prepare a VMPAGE disk for use in drive 1. 5. Using XDMS and UPDATE, enter all of your account numbers, (4 digits), into ACCTS along with a 10 character identifier name and a 50 character (max) full name/description. 6. Using XDMS and UPDATE, enter all of your department numbers, (3 digits max), into the OPSDEP file, along with a corresponding 35 character department name that you want to appear on each BUDGET page. 7. Using XDMS and UPDATE, enter the BASE program for update. For each department that you want to have a budget for, enter the DEPT number and a desired ACCT# number. Enter only the departments desired and accounts for each dept. This is the basis for your budget you must make sure you get all the entries correct. I suggest you leave the YEARLY field blank here and use the "I" command in UPDATE to repeat entries by department. As an example: If you had two departments numbered 130 & 600; and you wanted two accounts in dept 130 and 4 in dept 600; you must 6 total entries in the BASE database. Two will have dept number 130 and 4 will have dept 600. If both have an account number 7000 for Travel, then one of the entries for each dept must be account 7000. 8. Using XDMS and GENERATE command, run the BUILD program by entering:

XDMS:GENERATE BUILD<cr> This will merge all the ACCT# identifiers from the ACCTS file and will also add all the 12 months to create a new file called BUDBS. This is your main budget file. If for some reason you have forgotten to add an ACCT# in your BASE file for 1 or more departments, you can enter the DEPT and ACCT# info in BUDBS and the run GENERATE MERGE. This will add the correct account identifiers to the BUDBS file. I guarantee that you will need this option. The reason for these two activities is so that the identifier for each account will always show up the same no matter where you use an account number.

9. Using XDMS and GENERATE, the following program can be used to print a list of the departments and correspnding account numbers that you have entered. Run this program and check each entry to ensure that you have an identifier under the fieldname NAME. If you don't, then you have entered an account number into the file but you have failed to enter that same number into the ACCTS database. You must either correct your entry in the BASE database file or add the new account number into the ACCTS file.

Here is the program to check your entries: XDMS:P GENERATE <cr> FILE BASE<cr> PRINT DEPT,ACCT#,NAME<cr> SORT BY DEPT,SORT BY ACCT#<cr> ON DEPT SKIP 2<cr> MARGIN 5 TABSET 4 PAGE 60,6 PAUSE OFF<cr> END<cr>

This will print the desired proof list.

10. You are now ready to enter your own data and run the programs. Here are the only rules and limitations on their use:

RULES FOR DATA ENTRY

If you want the computer to "spread" the YEARLY amount evenly across each month, put the desired dollar value into the YEARLY field and ignore any fields with the name of the month. If you want to spread the data yourself and don't want the computer to do it, enter a 0 (zero) amount into the YEARLY field and then proceed to enter your own data into each of the months desired. The computer will total the amounts you enter into any of the months and place that value into the YEARLY field at the time of printout.

ORDER OF RUNNING PROGRAMS

You can run ACCTOT or ASSEMB at anytime. You can only run SUMMRY or PARENT after you have run at least one of the other two. This can be modified if you want to try your hand at changes.

LIMITATIONS

Data fields have the limitaions that have been identified before. The only other area that can be a problem is the size of the Integer fields that hold your monthly amount and also the YEARLY field. YEARLY can hold the number 99,999,999 without overflowing and truncating data. The monthly fields are 6 digit integers so they can only hold 999,999 before they overflow. These sizes were chosen carefully to enable all of the desired data to be printed in 132 columns. You may change them if you will exceed their limits but I can't imagine that you will need to.

PROGRAMS AND EXPLAINATION

BUILD.CTL

This program takes the BASE file with it's three fields, DEPT, ACCT#, YEARLY, and merges the corresponding 10 character account identifier name from the accounts file for each entry you have made in each department. This ensures that each account number gets the same name throughout the database. It also allows you a way to check and see if you have entered real account numbers. The resulting database created after BUILD is run, is the main file BUDBS. This file also includes the 12 monthly fields, ready for data input.

```
*********************************************************
*
*                BUILD.CTL
*
* REV 1.00        12-02-85        JA GERWITZ
*
*
*********************************************************

FILE BASE NAME BUDBS
SAVE DEPT ACCT#
MERGE ACCTS.NAME USING ACCT#
SAVE YEARLY
DEFINE JAN,I,6
DEFINE FEB,I,6
DEFINE MAR,I,6
```

```
DEFINE APR,I,6
DEFINE MAY,I,6
DEFINE JUN,I,6
DEFINE JUL,I,6
DEFINE AUG,I,6
DEFINE SEPT,I,6
DEFINE OCT,I,6
DEFINE NOV,I,6
DEFINE DEC,I,6
SORT BY DEPT SORT BY ACCT#
END
```

MERGE.CTL

This program does the same thing as BUILD except that it works on the file BUDBS. It is used after you have already run BUILD and have gotten to the point of adding a lot of account numbers into the main file. It should only be used for this purpose. This program leaves any data that you may have palaced into the YEARLY field and the monthly fields.

```
*****************************************
*
*              MERGE.CTL
*
* REV 1.00        12-02-85        JA GERWITZ
*
*
*****************************************

FILE BUDBS NAME BUDBS
SAVE DEPT ACCT#
MERGE ACCTS.NAME USING ACCT#
SAVE YEARLY ,JAN FEB MAR APR MAY ,JUN JUL AUG SEPT OCT NOV DEC
SORT BY DEPT SORT BY ACCT#
END
```

ACCTOT.CTL

This program generates the report TOTALS BY ACCOUNT. It should normally be run last after all your other work is done. The following is an explination of how it works. Line 16 thru 32 selects all of the records in the BUDBS file, which you want the computer to spread evenly across the 12 months, (done because YEARLY is not equal to zero). Line 40 thru 45 selects all the records which you have spread the monthly data, (done because YEARLY is equal to zero). Lines 48 thru 52 recombine the two groups of data into a temporary file named TEMPA. Lines 64 thru 72 merges the names of each of the operating departments into the file. Lines 93 thru 106 generates the actual report.

Several programming tricks are used to generate effects on the screen as well as manipulate data. See the section TRICKS later in the article.

```
*****************************************
*
*              ACCTOT.CTL
*
* REV 1.00        12-02-85        JA GERWITZ
*
*
*****************************************

FILE BUDBS
ECHO @1E @1B @59 ..... SORTING YEARLY RECORDS
RUN
*
*
FILE BUDBS NAME YEARAMTS
COUNT
COPY IF YEARLY <> 0
CALC YEARLY / 12 * 100 RND / 100 = REG1
CALC REG1 = ,JAN
CALC REG1 = FEB
CALC REG1 = MAR
CALC REG1 = APR
CALC REG1 = MAY
CALC REG1 = JUN
CALC REG1 = JUL
CALC REG1 = AUG
CALC REG1 = SEPT
CALC REG1 = OCT
CALC REG1 = NOV
CALC REG1 = DEC
RUN
*
*
FILE BUDBS
ECHO @1E @1B @59 ..... SORTING MONTHLY RECORDS
RUN
*
*
FILE BUDBS NAME MONTHS
COUNT
COPY IF YEARLY = 0
CALC JAN + FEB + MAR + APR + MAY + JUN + JUL + AUG + SEPT = REG1
CALC REG1 + OCT + NOV + DEC = YEARLY
RUN
*
*
FILE MONTHS [NO], YEARAMTS NAME TEMPA
ECHO @1E @1B @59 ..... SAVING RESPREAD BUDGET FILE
COPY
COUNT
RUN
*
*


FILE TEMPA NAME TEMPT
ECHO @1E @1B @59 .... CALCULATING TOTAL OF YEARLY AMOUNTS AND SAVING
DEFINE N N 1 FILL 1
SAVE YEARLY AS SUM-TOT
COUNT
SUM BY N
RUN
*
*
FILE TEMPA NAME TEMP
ECHO @1E @1B @59 ..... MERGING NAMES OF OPERATING DEPARTMENTS
DEFINE N,N,1 FILL 1
SAVE DEPT
MERGE OPSUMP,OP-DEPT USING DEPT
SAVE ACCT# NAME YEARLY ,JAN FEB MAR APR MAY ,JUN JUL
SAVE AUG SEPT OCT NOV DEC
COUNT
RUN
*
*
FILE BUDBS
ECHO @1E @1B @59 ..... MERGING TOTAL OF YEARLY DATA WITH RESPREAD DATA
RUN
*
*
FILE TEMP NAME TEMP
SAVE N,DEPT,OP-DEPT ACCT# NAME YEARLY
MERGE TEMPT,SUM-TOT USING N
SAVE JAN FEB MAR APR MAY ,JUN ,JUL AUG SEPT OCT NOV DEC
COUNT
RUN
*
*
```

```
FILE BUDBS
@ECHO @1E @1B @59 .... TOTAL RECORDS AVAILABLE FOR THIS REPORT -
RUN
*
*
FILE TEMP NAME TOTAL-BY
TITLE ===== TOTALS BY ACCOUNT FOR ALL DEPARTMENTS =====
SPACE PAUSE OFF MARGIN 2 PAGE 60,6
CACHE SUM-TOT
IF YEARLY <> 0
PRINT ACCT# NAME YEARLY
DEFINE PCT,D,3,2
PRINT JAN FEB MAR APR MAY JUN JUL AUG
PRINT SEPT OCT NOV DEC
CALC YEARLY / SUM-TOT * 10000 RND / 100 = PCT
SORT BY ACCT#
SUM BY ACCT#
TOTAL YEARLY PCT JAN FEB MAR APR MAY JUN JUL AUG SEPT OCT NOV DEC
END
```

ASSEMB.CTL

This program generate the report DEPARTMENT BUDGET. Most of the program is a duplicate of ACCTOT.CTL so that either can be run at any time. Lines 90 thru 115 actually generate the report.

```
*************************************************** *
*
*                ASSEMB.CTL
*
* REV 1.00          12-02-85          JA GERWITZ
*
*
*****************************************************


FILE BUDBS
ECHO @1E @1B @59 ..... SORTING YEARLY RECORDS
RUN
*
*
FILE BUDBS NAME YEARAMTS
COUNT
COPY IF YEARLY <> 0
CALC YEARLY / 12 * 100 RND / 100 = REG1
CALC REG1 = JAN
CALC REG1 = FEB
CALC REG1 = MAR
CALC REG1 = APR
CALC REG1 = MAY
CALC REG1 = JUN
CALC REG1 = JUL
CALC REG1 = AUG
CALC REG1 = SEPT
CALC REG1 = OCT
CALC REG1 = NOV
CALC REG1 = DEC
RUN
*
FILE BUDBS
ECHO @1E @1B @59 ..... SORTING MONTHLY RECORDS
RUN
*
*
FILE BUDBS NAME MONTHS
COUNT
COPY IF YEARLY = 0
CALC JAN + FEB + MAR + APR + MAY + JUN + JUL + AUG + SEPT = REG1
CALC REG1 + OCT + NOV + DEC = YEARLY
RUN
*
*
FILE MONTHS INCL YEARAMTS NAME TEMPA
ECHO @1E @1B @59 ..... SAVING RESPREAD BUDGET FILE
COPY
COUNT
RUN
*
*
FILE TEMPA NAME TEMPT
ECHO @1E @1B @59 .... CALCULATING TOTAL OF YEARLY AMOUNTS AND SAVING
DEFINE N N 1 FILL 1
SAVE YEARLY AS SUM-TOT
COUNT
SUM BY N
RUN
*
*
FILE TEMPA NAME TEMP
ECHO @1E @1B @59 ..... MERGING NAMES OF OPERATING DEPARTMENTS
DEFINE N,N,1 FILL 1
SAVE DEPT
MERGE OPSDEP,OP-DEPT USING DEPT
SAVE ACCT# NAME YEARLY JAN FEB MAR APR MAY JUN JUL
SAVE AUG SEPT OCT NOV DEC
COUNT
RUN
*
FILE BUDBS
ECHO @1E @1B @59 ..... MERGING TOTAL OF YEARLY DATA WITH RESPREAD DATA
RUN
*
*
FILE TEMP NAME TEMP
SAVE N,DEPT,OP-DEPT ACCT# NAME YEARLY
MERGE TEMPT,SUM-TOT USING N
SAVE JAN FEB MAR APR MAY JUN JUL AUG SEPT OCT NOV DEC
COUNT
RUN
*
FILE BUDBS
ECHO @1E @1B @59 .... TOTAL RECORDS AVAILABLE FOR THIS REPORT -
RUN
*
*
FILE TEMP NAME T-BUDGET
SPACE
COUNT
HEADING
                                                    X Y Z  Budget


==============================
DEPT NAME: <OP-DEPT>
==============================

END
CACHE OP-DEPT
CACHE SUM-TOT
IF YEARLY <> 0
PRINT DEPT ACCT# NAME YEARLY
DEFINE PCT,D,3,2
PRINT JAN FEB MAR APR MAY JUN JUL AUG SEPT
PRINT OCT NOV DEC
CALC YEARLY / SUM-TOT * 10000 RND / 100 = PCT
SORT BY DEPT SORT BY ACCT#
SUBTOTAL YEARLY PCT JAN FEB MAR APR MAY JUN JUL AUG SEPT OCT NOV DEC
TOTAL
ON DEPT EJECT
PAGE 60 6
PAUSE OFF
MARGIN 0
END
```

PARENT.CTL

This program can only be run after you have run ASSEMB.CTL or ACCTOT.CTL. As mentioned before, it is very usefull in the budgeting process because it allows you to see the source department for each account.

This is usually needed when your budget comes out too high and needs to be cut, (ever seen a budget too low?).

```
*************************************************
*
*                PARENT.CTL
*
* REV 1.00         12-02-85        JA GERWITZ
*
*
*************************************************


FILE TEMP NAME TOTAL-BY
TITLE ====== ACCOUNTS FROM PARENT DEPARTMENTS ======
SPACE PAUSE OFF MARGIN 2 PAGE 60,6
CACHE SUM-TOT
PRINT ACCT# NAME DEPT YEARLY
DEFINE PCT,D,3,2
PRINT JAN FEB MAR APR MAY JUN JUL AUG
PRINT SEPT OCT NOV DEC
CALC YEARLY / SUM-TOT * 10000 AND / 100 = PCT
SORT BY ACCT# SORT BY DEPT
SUBTOTAL YEARLY PCT JAN FEB MAR APR MAY JUN JUL AUG SEPT OCT NOV DEC
ON ACCT# SKIP 2
TOTAL
END
```

SUMMARY.CTL

This is the other program that you can only run after ASSEMB.CTL or ACCTOT.CTL. This program generates the total by department grouping of budget data. There are several unique items in this program that you should note since you may not want them.

a). In lines 47 thru 54, I pull the department 245 costs out of the 200 department summary. This is because my company identifies this cost as a seperate item. If you don't want it this way, delete lines 47 thru 54 and line 28. b). In lines 88 thru 95, I pull out ACCT# 7401, Interest Expense, from the department 500 summary. If you don't care to do this, delete lines 88 thru 95 and line 59. c). If you take either option a or b above, don't forget to drop the appropriate "INCL" statement and file name in lines 98 and/or 99.

Lines 98 thru 108 generate the sum of each department expenses. Some of you will notice that this format is part of a pro forma expense statement. That is why I wrote SUMMARY the way I did.

```
*************************************************
*
*                SUMRY.CTL
*
* REV 1.00         12-02-85        JA GERWITZ
*
*
*
* FILE TO GENERATE THE SUMMARY AMOUNTS
* FROM THE TEMP FILE, AFTER "ACCTOT.CTL"
* OR "ASSEMB.CTL" GENERATES THE "TEMP" FILE.
*
*************************************************


FILE TEMP NAME SUM100
CACHE DEPT
IF DEPT > 99, IF DEPT < 200
DEFINE NAME,A,35 FILL Sales Expense
DEFINE DEPTS,N,4 FILL 100
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM200
CACHE DEPT
OR DEPT <> 245
IF DEPT > 199, IF DEPT < 300
DEFINE NAME,A,35 FILL Production Costs
DEFINE DEPTS,N,4 FILL 200
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM305
CACHE DEPT
IF DEPT > 299, IF DEPT < 400
DEFINE NAME,A,35 FILL Warranty Cost
DEFINE DEPTS,N,4 FILL 305
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM245
CACHE DEPT
IF DEPT = 245
DEFINE NAME,A,35 FILL Service Costs
DEFINE DEPTS,N,4 FILL 245
SAVE YEARLY
SUM BY DEPTS
RUN
*
*


FILE TEMP NAME SUM500
CACHE DEPT
IF ACCT# <> 7401
IF DEPT > 499, IF DEPT < 600
DEFINE NAME,A,35 FILL General and Administrative
DEFINE DEPTS,N,4 FILL 500
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM600
CACHE DEPT
IF DEPT > 599, IF DEPT < 700
DEFINE NAME,A,35 FILL EDP Dept
DEFINE DEPTS,N,4 FILL 600
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM700
CACHE DEPT
IF DEPT > 699, IF DEPT < 800
DEFINE NAME,A,35 FILL R and D
DEFINE DEPTS,N,4 FILL 700
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
FILE TEMP NAME SUM501
CACHE DEPT
IF DEPT = 501, IF ACCT# = 7401
DEFINE NAME,A,35 FILL Interest Expense
DEFINE DEPTS,N,4 FILL 501
SAVE YEARLY
SUM BY DEPTS
RUN
*
*
```

```
FILE SUM100 INCL SUM200 INCL SUM305 INCL SUM245 INCL SUM500
INCL SUM700 INCL SUM501 INCL SUM400
SPACE
TITLE ------ COST SUMMARY BY DEPARTMENT TOTALS ------
NAME SUMRY
PRINT DEPTS.NAME.YEARLY
TOTAL YEARLY
PAUSE OFF
PAGE 60.6
MARGIN 15
END
```

TRICKS

There are a number of "tricks" that I have used, to do some of the things in these programs. Since I have a great deal of time invested in learning them, I would like to pass them along to you to shorten your learning cycle.

1). In several of the programs you will see the statement: "ECHO @1E 1B 59 ..... MESSAGE" This statement sends a home up, clear screen to my QVT-102A terminal and then prints the MESSAGE on the screen. You may use multiple ECHO commands if you need to.

2). I have found that in the version of XDMS that I have (1.1), an ECHO statement in a file with an IF statement will not work. To get around this, I put ECHO statements in seperate program segments as you see in lines 11, 12, and 13 in the ACCTOT.CTL program.

3). In order to calculate the percentage called PCT in the various reports, I needed to have the grand total of all expenses from the field YEARLY from all entries. I then needed to merge this value back into the file. The way I did this is shown in lines 55 thru 61 of ACCTOT.CTL. These lines of code save the grand total of YEARLY in a field named SUM-TOT of a file named TEMPT.DMS. I used another trick here to prepare a way to merge this data back later and also allow me to sum it in line 60. This is done by defining a new field "N", as (N)umeric of size 1 and FILLing it with a value of 1. This field will be used later for merging the data. In line 66 of ACCTOT.CTL, I add the same "N" field to the database. Then in line 82, I use the N field in the main file and the N field in the TEMPT.DMS file to merge SUM-TOT into all records. The value of SUM-TOT is then used later in the program to calculate PCT for each entry. Programs like this take longer to run than a spreadsheet would, but you have the advantage of unlimited records.

4). Please note the "OR" statement in line 28 of SUMMARY.CTL. If you read your XDMS manual v-e-r-y carefully, it is clear that "IF statements following OR statements are taken as logical AND criteria". This is demonstrated in line 28.

5). Use of the "FILL" command is demonstrated in several of the programs, especially SUMMARY.CTL.

6). These programs represent a good example of the power of a database as opposed to a spreadsheet. For very large jobs a database has certain advantages over a spreadsheet. I believe spreadsheets like DYNACALC are more flexiable and results are obtained quicker if your job is small, (ie. less programming).

7). Line 102 in the ACCTOT.CTL program correctly rounds the PCT result so that errors will not accumulate. The " * 10000 RND / 100" gives a correct result in the second decimal place. This was satisfactory for me, even though the sum of PCT can be 100.1 or 99.9.

8). The SHOW.CTL program is strictly an example of how to use the STRING command to seperate out a field and do something usefull. You could easily devise a set of ACCT#'s that could be broken down into sub-accounts etc. I added this strictly as an example.

FILE DEFINITIONS

This is what the file definitions should look like:

```
BASE              FILE--- PRINT--
# GRP/FLD  FMT    OFS LEN OFS LEN
1 BASE     G,3
  DEPT     N,3     0   2   0   3
  ACCT#    A,4     2   4   5   4
  YEARLY   I,8     6   4  11   9
                  ---------------
TOTAL RECORD LENGTH:  10      20

ACCTS             FILE--- PRINT--
# GRP/FLD  FMT    OFS LEN OFS LEN
1 ACCTS    G,3
  ACCT#    A,4     0   4   0   4
  NAME     A,10    4  10   6  10
  DESC     A,50   14  50  17  50
                  ---------------
TOTAL RECORD LENGTH:  64      67

OPSDEP            FILE--- PRINT--
# GRP/FLD  FMT    OFS LEN OFS LEN
1 OPSDEP   G,2
  DEPT     N,3     0   2   0   3
  OP-DEPT  A,35    2  35   5  35
                  ---------------
TOTAL RECORD LENGTH:  37      40

BUDBS             FILE--- PRINT--
# GRP FLD  FMT    OFS LEN OFS LEN
1 BUDBS    G,16
  DEPT     N,3     0   2   0   3
  ACCT#    A,4     2   4   5   4
  NAME     A,10    6  10  11  10
  YEARLY   I,8    16   4  22   9
  JAN      I,6    20   3  32   7
  FEB      I,6    23   3  40   7
  MAR      I,6    26   3  48   7
  APR      I,6    29   3  56   7
  MAY      I,6    32   3  64   7
  JUN      I,6    35   3  72   7
  JUL      I,6    38   3  80   7
  AUG      I,6    41   3  88   7
  SEPT     I,6    44   3  96   7
  OCT      I,6    47   3 104   7
  NOV      I,6    50   3 112   7
  DEC      I,6    53   3 120   7
                  ---------------
TOTAL RECORD LENGTH:  56      127
```

OS9 SPRING COMDEX '86

OS9 was much in evidence at Spring COMDEX '86 held in Atlanta, April 28-31, '86. First at a formal news conference where several important new OS9 developments were announced between Microware and new applications vendors.

One of the major vendors of new OS9 products is MicroTRENDS, Inc. MicroTRENDS introduced several new and significant products for OS9.

The most significant, in our view, was the demonstration of OS9 running, full bore, on the Atari 68000 systems, both the ST-520 and ST1040, at the Atari booth. Two terminals were assigned to the system (an ST-1040) and several popular OS9 software packages were demonstrated, including DynaCalc. I was surprised to see that the port was done so well. The main monitor was the Atari RGB color monitor, with different color backgrounds the resolution was very impressive! The second monitor was a Wyse-50 mono CRT terminal, the resolution was equally good.

The port of OS9 to the Atari was done by TLM SYSTEMS. TLM are the same folks who developed the 68000-OS9 for the IBM PC. Their experience with porting OS9 seems to have paid off in what appears to be a real winner. OS9 for the Atari will sell for $295.00 and includes BASIC09, according to the information we received. We will have an indepth review of OS9 on the Atari ST-1040 in an upcoming issue of 68 Micro Journal.

Another new product to the OS9 community, but a well known product in other circles, was VOLKSWRITER Deluxe. This is a popular word and text processor and formatter. It is very complete in it's range of functions and includes drivers for most all popular printers, including true proportional spacing. Also it is compatible with the popular 1-2-3, SuperCalc and other data base and spread sheet programs. Many of which will be made available for this OS9 system. Price was quoted at $199.00.

KnowledgeSet announced their KRS for CD-ROM available for the OS9 operating system. Included is the complete "Grolier Electronic Encyclopedia". OS9 which is the official system for the new Phillips-Sony CD-I supports such products. The first KRS is for the "Jonathan" 68000-OS9 card system for the Apple ||. The KRS is a complete search and retrieval application designed specifically for CD storage devices.

Also demonstrated at the MicroTRENDS booth was an image capture and display system. This video system was very impressive in the resolution of it's digital video reproduction. Named "The ColorCatcher" it certainly does that and more. Essentially it captures a real-time image from an NTSC video source then internally compresses it from an approximate 1 Mbyte RAM area to about 20K.

Capture rate and compression is approximately 3 seconds and display is 1 second. This includes the RAM replay expansion. Inputs and outputs are standard NTSC RS-170A. Pixels in graphics mode are 768 horz. by 488 vertical, 256 grey levels, 16 million color scales.

These, and other new products for OS9 are to be reviewed indepth in the near future. Due to time constraints, this preliminary report is to let you know what is coming. And of course let you in on the new and exciting OS9 goodies we saw at Spring Comdex '86. **OS9 has certainly become an industry standard.** Who knows what is next from Microware...well, actually I do, I think...but some of it has to wait. I can tell you this however, they are fast becoming a real force in the computer world. We are proud of our association with them, from day one!

By the way. This is the first output from our new Apple LaserWriter. Using STYLO. Yep, thats right, that STYLO, and we have STYLO running on a 68000 Apple Mac 512K (which is for sale at a good price - call if interested). As we get our porting straightened out, more and more will be done on this system. We still are sticking to our original promise - to publish 68 Micro Journal with "*OUR*" computer driven systems. This is just another step forward.

Just thought you might like to know.

something new.....

As you might have noticed, we are looking better. I have tried over the past 8 plus years to keep 68 Micro Journal a magazine that you could relate to...We must be doing something right, as we are now the third oldest computer magazine still going! In all there has been over 500 computer magazines, in the past 10 years or so. Of all those, large and small, we survive! I think it is because we have listened to you and keep pace with our market. So we will in the future, leaving no one of you behind, or forgotten.

We could have been fully typeset from day one, but I wanted to do it with our own type computers, we did.

Now we have a Mac™ and the new LaserWriter™, not the top of the line typesetting, but sufficient for our purposes. Also it is driven by 'our' type CPU. So I feel we have kept the faith.

Especially as we are using a special software package to drive it, from our old friends Great Plains. Their new STYLO FORMATTER for the Mac. Also it will drive our 100 grand typesetter, if we have the need. How 'bout that?

Will keep you posted as we progess. And if you now send in your articles in STYLO format, they will look better! Most already are, but I just wanted you to be aware.

We are also using a very nice package called PageMaker. As many of you require professional looking typeset manuals, catalogs, forms, etc. we can help with some experienced advice. Give me a call. We have tried about all the 'Desktop Publishing' software packages and I can sure save you some hard times and lots of bucks.

I am asking you to let me know what you think about our new format. Do you want it in 2 or 3 column format?. Or shall we mix it up? Also what type style do you find best for: 1. text materials and 2. source listings.

Thanks, and let me know. After all, we do listen. And please don't gripe, if you didn't respond.

DMW - - -

OS-9 *User Notes*

By: Peter Dibble
19 Fountain Street
Rochester, NY 14620

The Best of Times

No question about it, these are great days to be involved with OS-9/68K! CD-I looks like it's getting an excellent reception. The PC types (like Microsoft) seem worried; perhaps even they see OS-9 coming out of obscurity. The OS-9 port to the Atari ST has been announced for availability in June. Brian Lantz, President of the OS-9 Users Group, has tried it and pronounced it good (he said it more strongly than that). Ports to the Amiga and even the Macintosh are in the works and due out in the fall. Maybe the flood of news about OS-9 caught Byte's attention. In May, Byte held a conference on OS-9/68K on BIX, their teleconferencing system.

My new computer is still making me happy. I have good/bad news from Gimix. The good news is that they are enhancing the Micro-20, (Mustang-020™ as well?) with some interesting boards for the I/O connector. The bad news is that the graphics board is waiting for a new chip set from Hitachi. It won't be available this summer.

The I/O boards from Gimix are an interface to the Motorola I/O channel, and two different networking interfaces. The I/O channel is the high-speed I/O bus that Motorola uses with the VME bus. It is widely supported, so this will give Micro-20 users many sources for I/O cards.

Ed's Note: See article July '86 68 Micro Journal

One networking interface for the Micro-20 is relatively slow (I don't remember the number), and will use one of the four ports on the serial attachment. The other networking interface is Arcnet, which is a widely-used networking standard developed by Datapoint. The important feature of the Arcnet interface is speed -- hard disk speed. A network of Micro-20s (Mustang-20s) using either of these interfaces will be something special. If someone comes up with either a faster or a less expensive board to hook into the network it will be even more exciting.

The family of OS-9/68K ports to popular machines is the work of TLM. They have had a 68000 board with OS-9 on the market for the IBM PC for a while now. I guess they liked that so well that they decided to get deeply involved with OS-9. In any case the Atari ST should be a fine low-end OS-9 machine. The special features of the machine aren't supported ... yet. I understand that support for the graphics, and sound hardware are in the works, and there's already support for hard disks.

Volkswriter Deluxe is coming for OS-9/68K. I haven't been able to find out whether it will only be for the Atari or will be available for all of us. Let's hope.

I've been working on converting public domain Unix programs for my system. It's easy. The biggest project is a conversion of Hack. Hack is an elaborate interactive game, and a very large program. I think it will come to about 256K including program and data.

I found out that the linker can't deal with a program this big. The problem is that the process descriptor has slots for pointers to only 32 blocks of allocated memory. The linker is written in C. When a C program mallocs memory it gets it from the system with the ebrk function from the C library. Ebrk in turn uses FSSrqMem to get 4096-bytes of memory from the system. It can do this 32 times before the process descriptor runs out of slots; that's enough for 128K. I was a little surprised when my two megabyte system ran out of memory.

It turns out that I had to increase the size of the blocks of memory that the linker, l68, requests. The variable that needs changing is _memmins. You can read about it in the description of the ebrk function in the C manual.

There is a fix for this problem. Microware doesn't distribute the source for l68, so we can't just go in and assign a new value to _memmins, but we can use the debugger. The value of _memmins is at an offset of 615A from the start of the module. It is set to 0000 1000. I reset it to 0000 2000. The sequence of commands goes something like this:

```
load l68
debug
l l68
c .e7+615a+2
        10 20
        00 .
        q
        save l68 .
        fixmod l68 -u
```

It might be good to increase the default stack size for l68 too while you're in there. I had trouble with the linker's stack overflowing. It's usually easy to deal with stack overflows; you just request more memory on the command line, but I couldn't find any way to increase the memory allocation of the linker without calling it explicitly. It is usually called by cc with plenty of stuff on the command line. I didn't want to worry about all that "stuff" so I just doubled its default allocation with debug and had no more trouble.

When the next release of C comes out there should be a command-line option for setting the minimum allocation. Until then consider the problem (and the fix) documented.

I chose the latest version of Hack to convert. It has many features, and, I believe, it's somewhat buggy. But if I get it converted it will please my wife mightily.

I did some smaller programs first. The most useful of the batch I ported was Micro-Emacs. It's a nice, emacs-like, editor. I'm using it to write this column. As public domain programs go it is excellent. It isn't perfect; for example, the word wrap mode deletes punctuation if it falls at the end of a line. It won't take me long to fix that problem and the few dozen others I've found, then I'll have an editor at least as good as any I've seen sold for OS-9.

I'm working up to TeX. TeX is a text formatting program by Knuth. It is probably the best text formatting program available for people who are willing to invest some effort in their documents. It is big and complicated, but they say that it is easily portable. We'll see.

I picked up some interesting facts from the BIX conference. One that makes me particularly happy is that the Atari OS-9 and languages may be reasonably priced. Not down to the CoCo level, and not settled yet, but it seems they want to be reasonable. TLM is using a non-standard disk format for the Atari. Do you suppose they are trying to prevent us from buying cut-rate Atari OS-9 software and using it on our systems?

The moderator of the conference, Jim Omura, posted a list of timings for OS-9/68k SVCs. They were measured by Microware. I don't think they are for the latest version of OS-9, but they are useful numbers. The column from here on is quoted from BIX which was quoting Microware's literature.

The following information has been provided by Microware:

System timing calls can be critical for real-time control situations. This table provides the time required to execute the most commonly used system calls for a 68000 processor based computer.

Name	Function	8 MHz	10 MHz
(execution time in microseconds)			
Task Management System Calls			
Fork	Create Process	3797	3038
Send	Send Signal	217	174
Wait	Wait for process to terminate	290	232
Sleep	Suspend process for set time	183	146
SPrior	Change process priority	165	132
Time	Read time of day with ticks	210	16
ID	Read process ID	152	122
Memory Management System Calls			
Mem	Request User Memory	173	138
SRqMem	Request System Memory	205	164
Link	Link to memory module	1545	1236
Unload	Release memory module	1625	1300
DatMod	Create data module	11378	9102
CpyMem	Copy memory (1 byte)	170	136
CpyMem	Copy memory (4K bytes)	4265	3412
File I/O System Calls			
Create	Create a new file	10 40	8672
Open	Open existing file	7465	5972
Close	Close file	2038	1630
Delete	Delete file	1334	10678
Write	Write binary data (1 byte)	1120	896
Write	Write binary data (1024 bytes)	6223	4978
Seek	Position file pointer	247	198
Read	Read binary data (1024 bytes)	40 5	3276
WritLn	Write text line (80 bytes)	690	552
ReadLn	Read text line (80 bytes)	1180	944
Sequential I/O System Calls			
Create	Create new pipe	5147	4118
Close	Close pipe	2367	1894
ReadLn	Read Text line (80 bytes)	445	356
WritLn	Write text line (80 bytes)	440	352
Internal Functions			
Task Switch	Change active tasks	260	210
Interrupt	Time to reach 1st instruction of interrupt service routine ('n' represents the polling sequence number of a driver on a particular vector. For example, the time to access the service routine of the 3rd drive on a shared vector at 10 MHz is 40+(14*3)=80 mu.)	50+17n	40+14n

This report gives representative timings for critical or commonly used OS-9/68000 system functions. It is intended to provide a general picture of OS-9's outstanding performance and capability in real-time applications.

Hardware Considerations:

It is difficult to predict in advance the exact performance of specific hardware. ... [omitting some commentary] Each memory wait state per address strobe will increase the basic execution timing by approximately 25 percent.

I/O device performance also varies considerably. Therefore, the I/O related system calls were measured using RAM based devices, specifically, virtual disk for file I/O and pipes for sequential I/O. Timings include execution time of the device driver routines.

Software Considerations:

The exact execution time of OS-9 system calls can vary considersbly due to several factors. These include the specific parameters passed to the system call, the status of system resources, and the device performance (for I/O functions). The timings shown are for system calls with typical parameters: actual timings may vary considerably.

System clock tick interrupts occur randomly with respect to system calls. This would add a random variance to some timing measurements. Therefore, the clock was not enabled during the timing measurments. The clock overhead is generally less than one percent of the system throughout, depending on the exact clock hardware and speed.

Testing methodology:

Timings were obtained using a software-readable hardware counter which counts actual CPU cycles. The results shown represent the total time from execution of the system call trap by the test program until control returns from OS-9 to the test program. The timings for interrupt services and task switching were not measurable by this method so they were computed manually based on the instruction execution timings as given in official Motorola data sheets.

The DYNACALC Spreadsheet
A Review

By: Peter Dibble

I can't think of the technical merits of DYNACALC for the 68K without having its price leap to the front of my mind. The price is absolutely absurd! If you pay $600 for this program you will be getting a VisiCalc-class program for Symphony prices.

It is true that DYNACALC is the only competent spreadsheet program for OS.9/68K, but for $200 more I bought a lap computer with Calc-To-Go, Wordstar-To-Go, a communication program, and a stupid database. Several hardware vendors have bundled DYNACALC into similar packages at reasonable prices, but Computer Systems Center evidently isn't interested in quantity-one sales. We are a small market; vendors need to charge us high prices to recover their costs with small volume; I understand that. This, however, is unreasonable. OK, now I've gotten that off my chest. Let's consider the program.

Overview

All the major spreadsheets come from big companies. They were written by teams of programmers. Scott Schaeferle wrote DYNACALC, and is a real person, not a corporation. You might meet him at the next Microware Seminar. He clearly takes pride in his product. It works cleanly and fast, and he has enhanced it intelligently over the years. DYNACALC is not elaborate – it's not in the class of Symphony, or even Lotus 1 2 3 at all – but it does almost everything I've seen a spreadsheet do, and it does it fast.

What's fast? My Micro-20 running at 12.5 MHz does the Byte spreadsheet benchmark in about 0.43 seconds. Some spreadsheet on a PC took more than ten seconds. I admit that it isn't fair to compare a 68020 to an 8088, but I don't have a 68008 any more. DYNACALC for the 6809 is not what this review's about, but even on a CoCo it runs the benchmark faster than ten seconds. Spreadsheets are never fast enough, but it looks like it's hard to do better than DYNACALC.

The operation of DYNACALC is enough like VisiCalc that the DYNACALC manual refers the reader to a list of VisiCalc instruction books for additional help, but DYNACALC is not a VisiCalc clone (except in the sense that every spreadsheet program is a copy of VisiCalc).

The most obvious enhancement of this 68K version of DYNACALC over the version I have on my 6809 is the improved screen updating. Right behind that is the much larger limits on spreadsheet size. If I had enough memory, DYNACALC could handle a spreadsheet that was 18278 columns by 9999 rows.

There are more functions for financial calculations, and greatly expanded format definitions. If you spend the time to set the printer description file up, DYNACALC will format printed spreadsheets using bold and underlined characters in appropriate spots. It was easy to have it set my printer for 136-character lines before it starts printing.

Some Details

The DYNACALC manual is 65 pages long, and it certainly doesn't waste words. I can't even skim all DYNACALC's features in a review. I'm going to talk about just a few high points, and leave most of them out entirely.

The manual doesn't talk about screen management, but that is a major feature of this version of DYNACALC. If your terminal supports them, DYNACALC will insert and delete characters and lines to manage scrolling. It doesn't do this perfectly yet, but the improvement over 6809 DYNACALC is very noticeable. One note here: If your terminal, like mine, permits you to set your edit boundary, set it to lines. Otherwise character insertion and deletion will propagate from line to line causing the screen to sort of scroll sideways for character operations that should be isolated to one line.

Replication of cells is central to spreadsheet operation. Replication is like copying except that it can relocate addresses in the cells it is copying. If you replicate column A into column B, DYNACALC gives you the option of changing cell references in the copy to point to data in the copy or leaving them pointing at column A (the original). DYNACALC's replicate command is flexible. You can replicate from a range into a range. The ranges can be single cells, or general **blocks:**

```
C5...C10  data in a column
M20...ZZ20 data in a row
C5...ZZ20 a rather big block
If the source doesn't fill the destination it is used repeatedly.
```

Data can be moved even more generally than it can be replicated. Blocks can be moved (as they can be replicated). References to moved cells are adjusted automatically. You can also sort rows or columns by a specified key.

There is extensive support for different display formats. Evidently much feedback on previous versions was criticism about formatting tricks DYNACALC couldn't do. There should be much less of that criticism now. The formats for each cell may be selected from:

```
CCenter
DDefault
EEnhanced (Bold on screen and printer if possible)
GGeneral (Left justify labels, right justify numbers)
HHide (Don't display the contents)
LLeft justify
NCancels enhanced format
PProtected (contents of the cell can't be changed)
RRight justify
TTab is terminator for data entry.  The tab key will bring
the cursor to the next cell with tab format)
UUnprotect (cancels protected format)
ZZap (cancels tab)
*Continuous fills the cell with a repetitions of a single
character.  Saves memory over storing the whole cell of
repetitions).
#Value formats.

There are a variety of ways to display values.  These can be used in
combination!

BBlank if zero
DDump formula (show the formula instead of its result)
FFixed Decimal (1 to 9 places to the right of the decimal,
selectable)
GGeneral (vanilla floating point)
IInteger (rounds display to integer)
JJulian date (converts a julian date to month-day-year format)
PPlot
SScientific notation
TTrailing Minus
,Insert commas
$Display as dollars and cents
%Display as percent
(Put negative values in parentheses
```

There is some intelligence in the way DYNACALC displays dates. If the cell is narrow it uses a format like 5-5-86. If there is more space it uses 5-5-1986, with still more space it goes to May 5, 1986.

I'm not going to give an exhaustive list of the functions supported by DYNACALC. I'll just pick a few:

```
aSin. Cos. Tan. and inverses
eAverage. Max. Min. Count. Sum of the Squares. and
Standard Deviation all work on ranges of cells
oNet Present Value. Present Value. Future Value. Compound Interest.
Initial Investment: and Payment on a Loan.
rChoose. Index. and Lookup all find entries in tables
sCell can be used to treat a block of cells as a two
dimensional array.
tIf. And . and Or support conditional execution
```

Programs that want to run on all OS-9 systems need to be able to adapt to a wide variety of terminals and other displays. DYNACALC does this in a way that is relatively hard to configure, but powerful. It works better on my ANSI-standard terminal than any other program I have bought, but it took me three tries to get it right.

On the distribution disk is a directory with modules in assembled and text form for each of about a dozen terminals. These modules contain declarations for all the constants DYNACALC needs to drive a terminal. If you are lucky enough to have one of the terminals represented in this list (which is growing), and you can live with the assignments the module makes for each command key (<control>E to edit, <control>F to find, and so forth), you can move the correct module to your execution directory and start using the program. If none of the files meet your requirements, you find the text file closest to what you want and edit it.

The manual doesn't say anything about how to modify a terminal file. You have to rely on the excellent comments in the file. They are almost enough. I've spent a lot of time worrying about driving terminals, but I still made mistakes. It was hard to figure out what I had misunderstood when all I had to go from was characters spread all over the screen. Scott Schaefer did an exceptionally good job of using the assembler to trap errors in the terminal file, but he couldn't get them all.

Problems

The 68K version of DYNACALC is new, but the 6809 version has been around for a long time. This program shows that heritage. I found one problem with it, maybe two.

The problem is with the documentation for installation. The distribution disk includes a helps.dc file. It obviously needs to be put somewhere, because the help facility won't work without it. I tried putting it in /DD/SYS, /DD, /D0/SYS, /D0, /H0/SYS, /H0, and the current directory. Finally I put it in the execution directory, which turned out to be the right place. It's not intuitive to me to put a help file in the execution directory (though it does turn out to be a module). I've already complained about the lack of documentation for the terminal description file. I wish the manual had been a lot more explicit about the whole installation process.

I feel discontented with DYNACALC. It's the best spread sheet for OS-9, and even the 6809 version is better than anything for the PC in some ways. But, it's for all purposes the ONLY spread sheet for OS-9. We can't choose a spread sheet with all the bells and whistles like Framework or Symphony. It isn't really fair, but I want DYNACALC to measure up to the best in every way. So the tentative problem two is that DYNACALC doesn't beat the competition outside our corner.

Summary

It's a bit difficult to install DYNACALC, but once it's installed it does everything a spreadsheet should and does it fast. If you have enough memory it will handle vast amounts of data. It fits into the OS-9 environment well. I used it to crunch numbers for a house purchase with no trouble; all the financial equations I needed were built in. I used it to do some grading for a course I TA'ed for: no problem.

The 6809 version did the calculations for a car lease/purchase decision and kept my family and business budgets in excruciating detail. I have learned a lot by sorting budget data on various fields.

The 68K version has inherited and expanded on DYNACALC/6809's features (and its work). It is just a spreadsheet -- not a expert system for financial analysis, or an integrated office system -- but it's fast and complete. Within its limits DYNACALC is excellent.

+++

QPL

A Bold Step to Very-High-Level-Language

This is Part 3 of a series of articles about the QPL programming language. QPL is available for Flex-9 systems from Complier Products Unlimited, Inc. Phone (602) 991 1657.

In the first two articles in this series, we covered the basic language features, and discussed a feature new to most of you -- pattern matching.

In this part we will see how the use of QPL shortens the process of designing programs. We will also discuss user-written functions, and look at a text-formatting program.

PROGRAM DESIGN IN QPL

In writing a program, the steps a programmer takes are:
1. Requirements analysis - what will the program do.
2. Methods selection - how will the program do it.
3. Coding, test, completion.

In requirements analysis, we find out what the input and output data needs are, and user-selectable features. This step is fairly independent of the development language.

Methods selection consists of taking each requirement, and asking "how should I represent this data, and how should I manipulate it to get the required results?". One way of proceeding is to select a data representation, then 'walk through' a proposed processing algorithm. In this method the designer 'acts out' the functions of the program, substituting processing-in-his-mind for processing-in-the-computer.

In this 'acting out' method, the designer uses the human abilities of pattern recognition and selection to find and extract data relevant to each step of the process. Later the pattern recognition and selection processes are coded, often in obscure ways. It is here that the powerful pattern processing and conditional assignment functions in QPL can greatly simplify the final design stage, and the coding stage. The designer can use pattern matching and conditional assignment to recognize and extract the data in much the same way as when he 'hand executed' the program.

Now, to continue the description of the QPL language, we will look at subroutines, or as they are known in QPL, 'FUNCTIONS'.

USING FUNCTIONS

When do we use functions? They are used for two different purposes; to encapsulate a code sequence, and to provide a reusable code sequence.

To use functions, a programmer needs to know these things; How to declare the function, how to call it, and how to pass arguments. Like the 'C' language, QPL functions return a value, so this is a forth aspect of functions to look at.

FUNCTION DECLARATION & CALL

To declare a function which will perform the MOD operation (return remainder from division) we would write

```
FUNCTION(MOD,DIVIDEND,DIVISOR).
```

This creates a new function which acts like the built-in functions in most respects. The new function name is 'MOD'. It has two formal arguments 'DIVIDEND' and 'DIVISOR'. When we call the function MOD, we can substitute actual program variables for the formal arguments. For example, we could do a call like

```
INDEX = MOD(ARRAY_SIZE,SELECTOR).
```

In this example, the program variables 'ARRAY_SIZE' and 'SELECTOR' are the 'actual arguments'. Note that in the above example, we are using the value returned by the function, and assigning it to 'INDEX'. Figure 2 shows the code for the MOD function. It illustrates an important feature of QPL; that numbers and strings are treated as the same 'type'. That is, we may do string functions such as concatenation and pattern matching on numbers.

ARGUMENTS:

Function arguments are not type-declared, so any kind of value may be passed in the arguments. In the example above, ARRAY_SIZE may be an integer number, a real number, or a string. We may even pass an array into a function; however the function must know the array dimensionality.

QPL arguments are passed by reference. This means that like 'VAR' arguments in Pascal, changing the value of a formal argument will change the value of the actual argument. To clarify this concept, refer to the examples in Figure 1. Figure 1 shows two simple examples. The first shows how changing the value of a formal argument changes the value of the corresponding actual argument or variable which was passed to the function. The second example in Figure 1 shows how to avoid this effect by copying the argument value to another variable inside the function.

FUNCTION VARIABLES:

How many times have you been doing program development and decided that you needed to be able to access a local variable inside a Pascal procedure? So you move that variable to the global data declaration level and find its name conflicts with another one. Now you need to make name changes in the procedure. If the changed name is used in a nested procedure, the need for a simple change can become a half-hour project. Several compilations later you are ready to try your new code. In QPL we shorten this process since all variables are global and static. That means that all QPL variables will have a defined value at all points in execution.

PRIVATE VARIABLES

To gain some privacy with variable names, use the following simple naming policy: In each function, name all variables using a compound name which begins with the function name. For example, in the MOD function, if I need a variable named 'QUOTIENT', name it MOD_QUOTIENT. Now we don't have to worry that some other piece of code will accidentally write to our semi-private variable. The problem of accidental writes to variables is most common in languages which restrict the length of variable names to something short, like 6 characters. QPL allows variable names as long as you want, and all characters are significant.

RETURNED VALUES

User defined functions must end with a return statement. The form of the return is

RETURN(SOMETHING)

The 'SOMETHING' may be a variable value, or NULL. Whatever is returned, the function is replaced by the value returned. This is how the function 'MOD', discussed above, returns its value. The returned value may be assigned to another variable, or not, as desired.

TEXT-FORMATTING EXAMPLE PROGRAM

QPL is well suited to programs which manipulate text. For an example program, we will look at a text formating program, shown in Figure 3.

This formatter will produce 'ragged right' format output. It will read from an input file named 'TEXT.TXT', and will have a fixed output line length of 55 characters. Unlike some text formatters, the need for commands is minimized by building in more intelligence. For example, some text formatters require a '.sp' command to create a blank line. This simple formatter knows a blank line when it sees it and acts accordingly.

The text formating program consists of a 'main loop' and a single function. The function is called only from one place in the main, so it is an encapsulation type function.

Referring to the program in Figure 3, you see that the first step is to develop the patterns needed. In this case, we have three patterns for extracting words from sentences. The first word-extraction pattern is FIRST_PAT, which extracts the first word in a line. The second is MID_PAT, which extracts words which are not the first or last word. The last is LAST_PAT, which extracts the last word in a line, and allows the program to recognize that the text line has been consumed. In addition, we need a pattern which will recognize the case where a line consists of a single word. All these patterns search for a space in one way or another, since spaces delimit words.

Let's look at the pattern 'MID_PAT', in some detail. This is a somewhat complicated pattern, which does a lot for us. The first part of MID_PAT has a special pattern SPAN(" "), which scans over any spaces before a word, and this is followed by conditional assignment to 'SPACE'. That means that if there are any spaces before the word, SPACE will contain them after a MATCH is performed. Next, is a concatenated BREAK(" ") pattern which will continue the scan until the next space is found. This is followed by conditional assignment to WORD, so the word found will be copied to WORD. Last, there is another special pattern LEN(1500) which will match any remaining text in the line and copy it to BUF0. Now, if we use this pattern in a MATCH statement with BUF0, such as MATCH(BUF0,MID_PAT), then all these pieces will be extracted from BUF0, and the remaining text assigned back to BUF0. This kind of multi-part pattern gives QPL programs much power for little code.

The program design calls for the main program to read in a line of text to BUF0, and if it is not a blank line, call the formating function 'FORMAT'. No argument is passed, since the FORMAT function has access to BUF0 directly, as in Basic. The FORMAT function builds up a line of words in 'OUT' and when addition of another word would overflow the length limit, the line is printed out. When the input line in BUF0 is exhausted, FORMAT returns to the caller (the main program). The main program determines if it has read in a blank line, and if so, prints out the remaining text in OUT.

Figure 4 shows a sample input text and the formatted output text. If we wanted to get right-justified text, we would have the FORMAT function call another function 'JUSTIFY' rather than printing out the line.

This simple formatting program is not a commercial grade text processor; however it may be expanded easily with a command recognizer and right-justification capabilities. Excluding comments and blank lines, this text formatter has only 39 lines of code.

FIGURE 1 Use of pass-by-reference in functions.

```
* MAIN PROGRAM TO DEMONSTRATE THE USE OF ARGUMENT
* PASS-BY-REFERENCE AND VALUE RETURNED

 COUNT = 12
* In the following call to the function INC,
* the value of the argument (COUNT) will be
* changed by the call-by-reference feature and
* the fact that the function INC assigns a value
* to its formal argument (NUMBER).

 INC(COUNT)
* Count now has a value of 13.

* Next, we will call the function 'INCR' which
* does not change the value of the actual argument.
* because no assignment is made to the formal
* argument in the function 'INCR'

 COUNT = INCR(COUNT)
* COUNT now has the value of 14

*END OF MAIN PROGRAM AND BRANCH TO END OF LISTING
 :(END)
* ------------------------------------------

FUNCTION(INC,NUMBER)
* Action: Increments a number by changing
*       the value of the argument passed in.

 NUMBER = NUMBER + 1
 RETURN(NULL)

* ------------------------------------------

FUNCTION(INCR,NUMBER)
* Action: Returns a value which is 1 higher
*       than the value passed in.

 INCR_NUM = NUMBER + 1
```

```
 RETURN(INCR_NUM)
* ----------------------------------------

END
```

FIGURE 2 MOD FUNCTION

```
 FUNCTION(MOD,DIVIDEND,DIVISOR)
* Action:  Returns remainder of DIVIDEND / DIVISOR.
*         Does not change values of either
*         actual argument.

* First we create a pattern which will determine
* if the result of the division contains a decimal
* point, and if so, will copy the integer part
* to the variable 'MOD_QUO'.
 MOD_PAT = BREAK(".") . MOD_QUO

* Next, we initialize the remainder to 0,
* and do the division.

 REMAIN = 0
 MOD_QUO = DIVIDEND / DIVISOR

* Now, we determine if the division resulted
* in no fractional part.  If there was no
* fraction, then return 0.

 MATCH(MOD_QUO,MOD_PAT)    :F(MOD_END)

* We know that there was a fractional result,
* so compute the remainder and return it.

 REMAIN = DIVIDEND - (MOD_QUO * DIVISOR)
MOD_END  RETURN(REMAIN)

* ----------------------------------------
```

```
*****************************************
*       FIGURE 3
*    Text formatting program.
*
*****************************************

* Create the word-parsing patterns.
 FIRST_PAT = BREAK(" ") . WORD & LEN(1500) . BUF0
 MID_PAT = SPAN(" ") . SPACE & BREAK(" ") . WORD &
LEN(1500) . BUF0
 LAST_PAT = SPAN(" ") . SPACE & LEN(100) . WORD |
LEN(100) . WORD
 MULTI_PAT = SPLIT(" ")
* 'Hard code' the line width & file name.
 WIDTH = 55
 OPEN(1,"TEXT.TXT",GET,SEQUENTIAL)
READ_LOOP  STAT = READ(TEXT,TXT,BUF0)
* If the line just read in is not blank, THEN format it.
 IDENT(BUF0,NULL)                :S(IN1)
 FORMAT(NULL)                    :(CK_DONE)
* ELSE IF OUT is not null, print it out and print out a blank line.
* Also, clear OUT.
IN1  IDENT(OUT,NULL)             :S(IN2)
 OUTPUT = OUT
 OUT = NULL
* ELSE print out only a blank line.
IN2  OUTPUT = NULL
CK_DONE  EQ(STAT,8)      :S(FILE_END)F(READ_LOOP)

* ----------------------------------------
 FUNCTION(FORMAT,NULL)
* Repeatedly split off single words from input string,
* appending them to output string 'OUT' as long as output string
* length is less than WIDTH.  When word would make output string
* too long, print output string.  When input string is empty, return.
FORM_LOOP
 IDENT(BUF0,NULL)                :S(F_END)
* IF line contains no space, then WORD = LINE
 MATCH(BUF0,MULTI_PAT)           :S(FL5)
 WORD = BUF0
 BUF0 = NULL                     :(GOT_WORD)
FL5
* IF 1st char <> space, then use FIRST_PAT to get 1st word.
 MATCH(BUF0," ")                 :S(FL6)
 MATCH(BUF0,FIRST_PAT)           :(GOT_WORD)
FL6
* The text is either 'space-word-end', or 'space-word-space...'
* IF word_pat succeeds, text is 'space-word-space....'
 MATCH(BUF0,MID_PAT)             :S(GOT_WORD)
* ELSE text is 'space-word-end'.
 MATCH(BUF0,LAST_PAT)
GW2  BUF0 = NULL

GOT_WORD  CHR_CNT = SIZE(OUT) + SIZE(WORD)
* If the added word would overflow the line, THEN
 GE(CHR_CNT,WIDTH)               :F(FL4)
*    stick word back on input string.
 BUF0 = WORD & BUF0
*    Print out output string
 OUTPUT = OUT
*    Clear output string & goto FORM_LOOP
 OUT = NULL                      :(FORM_LOOP)
* ELSE (added word would not overflow output string)
*    append word to output string & goto FORM_LOOP
FL4  IDENT(OUT,NULL)             :S(FL3)
 OUT = OUT & SPACE & WORD        :(FORM_LOOP)
FL3  OUT = WORD                  :(FORM_LOOP)
F_END  SPACE = " "
 RETURN(NULL)

FILE_END
```

```
*****************************************
* FIGURE 4  Text input & output.
*    Text file to be formatted
*
*****************************************

This is a text file to test the text formatter named FORM.

It must work on simple text.
It has a fixed formatting line length of 55 characters.
FORM only does 'ragged right' formatting; however it would be simple
to add a function which would add spaces to right-justify the text.

We will see how this simple formatter works.

*****************************
*
*  Formatter output
*
*****************************

This is a text file to test the text formatter named
FORM.

It must work on simple text. It has a fixed formatting
line length of 55 characters. FORM only does 'ragged
right' formatting; however it would be simple to add a
function which would add spaces to right-justify the
text.

We will see how this simple formatter works.

-------------------------------------------------------
```

SPECIAL

Telex 5106006630
(615)842-4600
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

K-BASIC

K-BASIC under OS-9 and FLEX will compile TSC BASIC, XBASIC and XPC Source Code Files.

K-BASIC now makes the multitude of TSC XBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- useable, multi-precision, familiar Software is running under favorite Operating System!

$AVE $100

!!! SPECIAL ~~$199.00~~ $99.00 !!!

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi user macros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

With every development system you receive:

- ☐ A manual that makes sense
- ☐ A periodic newsletter
- ☐ Screen form language
- ☐ Report generator
- ☐ Menu system
- ☐ Query facility
- ☐ Set of utility programs
- ☐ Sample programs

For resale products, the run-time system is available at a nominal cost.

DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

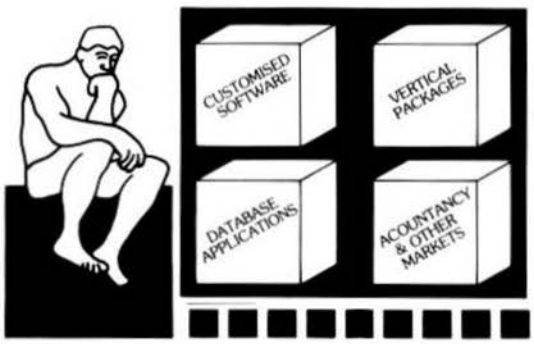
- ☐ alphanumeric
- ☐ integer
- ☐ floating point
- ☐ money
- ☐ date

DATA FILE STRUCTURE

- ☐ Packed, fixed-length records
- ☐ Money stored in lower currency unit
- ☐ Dates stored as integer day numbers

SCULPTOR

Facts Features



INDEXING TECHNIQUE

Sculptor maintains a B tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION

Input data may be validated at three levels.

- ☐ automatic by field type
- ☐ validation list in data dictionary
- ☐ programmer coded logic

ARITHMETIC OPERATORS

-	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
-	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to
< >	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

- ☐ Full date arithmetic
- ☐ Echo suppression for passwords
- ☐ Terminal and printer independence
- ☐ Parameter passing to sub-programs
- ☐ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

- ☐ Define record layout
- ☐ Create new indexed file
- ☐ Generate standard screen-form program
- ☐ Generate standard report program
- ☐ Compile screen-form program
- ☐ Compile report program
- ☐ Screen-form program interpreter
- ☐ Report program interpreter
- ☐ Menu interpreter
- ☐ Query facility
- ☐ Reformat file
- ☐ Check file integrity
- ☐ Rebuild index
- ☐ Alter language and date format
- ☐ Setup terminal characteristics
- ☐ Setup printer characteristics

SCREEN-FORM LANGUAGE

- ☐ Programmer defined options and logic
- ☐ Multiple files open in one program
- ☐ Default or programmer processing of exception conditions
- ☐ Powerful verbs for input, display and file access
- ☐ Simultaneous display of multiple records
- ☐ Facility to call sub-programs and operating system commands
- ☐ Conditional statements
- ☐ Subroutines
- ☐ Independent of terminal type

Sculptor for 68020 OS-9 & UniFLEX $995

MS DOS		**OS-9/UniFLEX**	**68000 UniFLEX**
PC DOS	**--$595 / $115** * **	**IBM PC Zenix**	**-- $995 / $175** * **	**Altos Zenix**	**-- $1595 / $265** * **
		MS DOS Network		**UNIX**	

*** Full Developement Package**
**** Run Time Only**

Full OEM & Dealer Discounts Available!

Sculptor is a Trademark of Microprocessor Developments Ltd.

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends–

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

ASSEMBLERS

ASTRUK09 from Southeast Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, CCF - $99.95

Macro Assembler for TSC -- The FLEX STANDARD Assembler.

Special -- CCP $35.00; F $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX.

FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F,CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F,CCP - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8

F,CCF - $98.00

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- (REAL ASSEMBLERS, NOT MACRO SETS) SPECIFY FOR 180X, 6502, 6800, 6801, 6804, 6805, 6809, Z8, Z80, 8048/51/85, 68000, modular, free standing cross assembler in "C", with LOAD/UNLOAD utilities and more. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text also.

FLEX, CCF, OS-9, UniFLEX assemblers each $50.00
any 3 objects or source (not 68XXX) - $100.00
Set of ALL objects $200.00 - source $300.00
UniFLEX 68000 $50.00 - source $1,000.00

XASM Cross Assemblers for FLEX from Compusense Ltd. -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.

Complete set, FLEX only - $150.00

CRASMB from Lloyd I/O -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sets, 80/85, Z-8, Z-80, TMS-7000 sets. Supports the target chip's standard mnemonics and addressing modes.

FLEX, CCP, OS-9 Full package -- $399.00

CRASMB 16.32 from Lloyd I/O -- Cross Assembler for the 68000.

FLEX, CCP, OS-9 $249.00

UTILITIES

Basic09 XRef from Southeast Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.

O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCP --- EACH Utility - 5" - $40.00, 8" - $50.00

DUB from Southeast Media -- A UniFLEX "basic" De-Compiler. Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.

U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F and CCP, U - $25.00, w/ Source - $50.00

SOLVE from Southeast Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95
* SPECIAL INTRODUCTION OFFER * $69.95

DISK UTILITIES

OS-9 VDisk from Southeast Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Oimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

-- Level I ONLY -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk.

OS-9 - $79.95

LSORT from Southeast Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and error messages.

OS-9 $85.00

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

HIER from Southeast Media - HIER is a modern hierarchal storage system for users under FLEX. It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. Using HIER a regular (any) FLEX disk (8 - 5 - hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution that all current large systems use. Each directory looks to FLEX like a regular file, except they have the extension '.DIR'. A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

Regular $139.95
* Introduction Special * $69.95

COPYMULT from Southeast Media -- Copy LARGE Disks to several smaller disks. FLEX utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation. Completely documented Assembly Language Source files included.

ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal NOT required. Allows reading TSC Mini-FLEX, SSB OOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F and CCF 5" - $50.00 F 8" - $65.00

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source (either BASIC or A.L. Source Code).

F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00
UniFLEX 68000 with complete Source - $100.00

X-TALK from Southeast Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Cirmx 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks) $99.95
X-TALK Software (2 disks only) $69.95
X-TALK with CMODEM Source Included $149.95

XDATA from Southeast Media -- A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.

U - $299.99

GAMES

RAPIER - 6809 Chess Program from Southeast Media -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point moving system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels).

F and CCF - $79.95

EDITORS & WORD PROCESSING

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.obj file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA
MasterCard

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .s .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95
OS-9 68K Version compiled with Source - $79.95

PAT from Southeast Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50
* SPECIAL INTRODUCTION OFFER * $79.95
SPECIAL PAT/JUST COMBO (w/source) FLEX $99.95
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K Version $249.00

Note: JUST is "C" source available for OS-9

CEDRIC from Southeast Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95
* SPECIAL INTRODUCTION OFFER * FLEX $69.95

BAS-EDIT from Southeast Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCF, STAR-DOS Regular $69.95
Limited Special Offer: $39.95

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

SPELLB "Computer Dictionary" from Southeast Media -- OVER 150,000 words! Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

CCF and CCO - $59.95, F or O - $79.95, U - $129.95

STYLO-PAK -- Graph + Spell + Merge Package Deal!!!

F or O - $329.95, U - $549.95

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809 and 68000.

F, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete 'item description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL-SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and maintaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, CCP - $198.00

PASC from Southeast Media - A Flex9 Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

FLEX $95.00

WHIMSICAL from Whimsical Developments -- Now supports Real Numbers. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9.

F and CCF - $195.00

KANSAS CITY BASIC from Southeast Media - Basic for Color Computer OS-9 with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCoOS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries.

F and CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

FLEX, CCF, OS-9 (Level II ONLY), U - $575.00

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F and CCF - $99.95

PASCAL Compiler from OmegaSoft (now Certified Software) -- For the PROFESSIONAL; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.

F and CCF - $425.00
One Year Maint. - $100.00
OS-9 68K Version - $900.00

K-BASIC from LLOYD I/O -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

FLEX, CCF, OS-9 Compiler with Assembler - $199.00

CRUNCH COBOL from Compusense Ltd. -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.

FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST MI. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

DATA-BASE ACCOUNTING

DATABASE - ACCOUNTING

XDMS from Westchester Applied Business Systems - Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new file Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

XDMS IV from Westchester Applied Business Systems - XDMS IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

XDMS IV - F, CCFSTAR-DOS, SK*DOS - $350.00
Upgrades to XDMS IV - $250.00

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)

CCD (32K Req'd) Obj. Only $49.00
CCF, Obj. Only $50.00 - w/source $99.00
F $99.00 - U, $100.00 - OS-9 $101.00
CCO, Obj. Only $50.00

DYNAMITE + from Computer Systems Center -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 CCO, Obj. Only $59.95
F, " " $100.00 O, " " $150.00
U, " " $300.00

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends~

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

BIT-BUCKET

By: All of us.....

MUSTANG-020™ Update

The MUSTANG-020™ still continues to be *#1* on our sales charts. Seems we will be having something new to tell you about the system every month or so.

IN THE PLANNING STAGE:

In the planning stage are the following:

- **Graphics Interface using the Hitachi ACRTC.**
- **A new Prototyping Board.**
- **Full SCSI interface (will support disconnect, reconnect, bus arbitration, etc.)**
- **Intelligent X.25 Adaptor.**

These new features will use the existing *I/O Expansion Connector*, available on all MUSTANG-020™ systems.

Currently we are beginning to receive a slightly better supply of 16 Mhz devices. This has been one of (practically the only real) slow-down of delivery dates. For the standard 12.5 Mhz systems, deliveries have been getting out of here in about 48 to 72 hours..after we do a sufficient *burn-in*. I insist on a complete burn-in; *quality control* is one area I insist is always on the front burner.

We will let you know, in these pages, as soon as any of the above planned items become available. If you need additional information, please give me a call.

DMW

- - -

Alan Shigemura
P.O. Box 23419
Honolulu, HI 96822

Dear Don,

I believe in keeping my letters short and precise. I've seen a lot of cheap IBM and IBM clone hardware come into the market. I wish someone would develop an adapter for these hardware so that I may be able to use these hardware on the S or SS-50 Bus.

Alan
Video & Memory

Ed's Note: Not enough interest shown, so far. Need at least 400 - 700 potential sales to make worth-while.
DMW

Werner F.W. Zychlinski
Pezelstr. 22
2820 Bremen 70
W.Germany

PL9 and FFT

In the April '86 issue of the 68' Micro Journal are some programs to calculate the Fast Fourier Transform from James H. Cross Jr. The programs are all in PASCAL language and can be adapted to other languages.

The Uhrich algorithm on page 45 and 46 is adapted to the PL9 language. The results are decripted below.

The result timings are shown in table 1. You can see that PL9 is very fast! But for my application, it's not fast enough. So I replace the internal Floating Point Package from PL9 with an package who is using the AM9511. The timing result with the same unchanged PL9 FFT program can we see in the row "+AMD FP". Now I replaced the whole "SCIPACK.LIB" library with an AM9511 library. The timing can we seen in row "+AMD SIN". The result with both the Floating Point Package and SCIPACK library with the AM9511 can we see in row "FP&SIN". The computation of an FFT with 1024 input data points is now 2.5 times faster than the original PL9 progrsm, and 8.9 times faster then the original PASCAL program in the 68' Journal.

But I think we can faster. So I write the complete "fastfourier" procedure in Assembler under full using the AM9511. This gives an dramaric improvement as you can see in row "+AMD FFT" ! The FFT under using the Uhrich algorithm on 1024 input data points is now 13.6 times faster the PL9 program and 50 times faster the original PASCAL program! With this speed and some hardware we can do an real time FFT analysis of audio frequency signals.

Table 1

N	LUC PASC	raw PL9	+AMD FP	+AMD SIN	FP&SIN	+AMD FFT
8	0.46	0.14	0.09	0.07	0.04	0.01
16	1.08	0.35	0.21	0.16	0.11	0.02
32	2.51	0.76	0.51	0.36	0.26	0.06
64	5.69	1.74	1.17	0.82	0.60	0.11
128	12.67	3.78	2.54	1.82	1.36	0.24
256	27.91	7.91	5.45	4.02	1.05	0.56
512	60.96	16.66	11.60	8.80	6.76	1.20
1024	132.16	35.49	24.55	19.15	14.81	2.64

where "LUC PAS" Luc Data PASCAL timing from April '86 page 46 of 68' Micro Journal
"raw PL9" is the original program adapted to PL9 Version 4.21,
"+AMD FP" is PL9 but the Floating Point computations are replaced by the AM9511,
"+AMD SIN" is PL9 with SIN and COS are replaced by the AM9511,
"+FP&SIN" The PL9 Floating Point Package, SIN and COS are replaced with the AM9511,
"+AMD FFT" The whole "fastfourier" Procedure was replaced by an assembler program who used the AM9511 for all integer and Floating Point computations.

The time is measured with an hardware timer from the procedure "inputfunction" to the procedure "outputresult". All values in seconds on an 2 Mhz 6809 with 4 Mhz AM9511.

Here are now the raw PL9 FFT program:

```
/*   FFT: The Uhrich Algorithm in PL9   */

GLOBAL BYTE DATA(1024):       /* Input data from A/D converter */
       REAL RESULT_re(1024),  /* Floating-Point result: real   */
            RESULT_im(1024);  /*           and imaginary part  */

INTEGER P2 1,2,4,8,16,32,64,128,256,512,1024,2048;     /* 2^X  */

INCLUDE TRUFALSE.DEF ;
INCLUDE IOSUBS.LIB ;
INCLUDE SCIPACK.LIB ;
INCLUDE REALCON.LIB ;
INCLUDE REALIO.LIB ;


PROCEDURE inputfunction(BYTE .DATA: INTEGER N): INTEGER I;
/*A 12.52 pulse is defined in this function*/
```

```
BEGIN
   I = 0 ;
   REPEAT
      DATA(I) = I ;
      I = I + 1 ;
   UNTIL I > N/8 ;
   I = N/8 ;
   REPEAT
      DATA(I) = 0 ;
      I = I + 1 ;
   UNTIL I > (N-1) ;
END;
ENDPROC ;

PROCEDURE fastfourier(BYTE    .DATA:       /* input data       */
                      REAL    .RESULT_RE,  /* Real result      */
                              .RESULT_IM:  /* imaginary result */
                      INTEGER N):          /* Number of points */

INTEGER   A,B,C,D,E,F,I,J,r,ln2),NZ,
          PNP2J,ln2)N2,NSTAGE,NP2I,NP2JI,P2J2:
REAL      M_re,M_im,W,wiew,cosw,arg,FN,
          TEMP_RE(1024),   TEMP_IM(1024);
BEGIN
   NSTAGE = 11 ;
   WHILE N<>P2(NSTAGE) NSTAGE = NSTAGE - 1 ;
   W = 3.1415925/FLOAT(N/2) ;
   N2 = N/2 ;
   FN = FLOAT(N) ;
   I = 0 ;
   REPEAT
      TEMP_RE(I) = FLOAT(DATA(I)) ;
      TEMP_IM(I) = 0.0 ;
      I = I + 1 ;
   UNTIL I>N-1 ;
   I = 1 ;
   REPEAT
      NP2J  = N/P2(J) ;
      NP2JI = N/P2(J-1);
      P2J2  = P2(J)/2 ;
      I = 0 ;
      REPEAT
         ln2j   = I*NP2J ;
         arg    = W*FLOAT(ln2j) ;
         cosw   = COS(arg) ;
         niew   = SIN(arg) ;
         F      = I*NP2JI ;
         PNP2J  = F + NP2J;
         ln2jN2 = ln2j + N2;
         r = 0 ;
         REPEAT
            A            = r + ln2j ;
            B            = r + F ;
            C            = r + PNP2J ;
            E            = r + ln2jN2 ;
            M_re         =  TEMP_re(C)*cosw ;
            M_im         = -TEMP_re(C)*niew ;
            RESULT_re(A) = TEMP_re(B) + M_re ;
            RESULT_re(E) = TEMP_re(B) - M_re ;
            RESULT_im(A) = TEMP_im(B) + M_im ;
            RESULT_im(E) = TEMP_im(B) - M_im ;
            r = r + 1 ;
         UNTIL r>(NP2J-1) ;
         I = I + 1 ;
      UNTIL I>(P2J2-1) ;
      r = 0 ;
      REPEAT
          TEMP_re(r) = RESULT_re(r) ;
          TEMP_im(r) = RESULT_im(r) ;
          r = r + 1 ;
       UNTIL r>(N-1) ;
       J = J + 1 ;
   UNTIL J>NSTAGE ;
   r = 0 ;
   REPEAT
      RESULT_re(r) = RESULT_re(r)/FN ;
      RESULT_im(r) = RESULT_im(r)/FN ;
      r = r + 1;
   UNTIL r>(N-1) ;
   END ;
ENDPROC ;


PROCEDURE MAG(REAL x,y) ;
ENDPROC REAL, SQR(x*x+y*y) ;


PROCEDURE outputresult(BYTE    .DATA:       /* input data       */
                       REAL    .RESULT_RE, /* Real result      */
                               .RESULT_IM: /* imaginary result */
                       INTEGER N):          /* Number of points */
INTEGER I ;
BEGIN
   I = 0;
   REPEAT
      CRLF ;
      PRINTINT(I);           SPACE(4);
      FPRINT (DATA(I));      SPACE(2);
      FPRINT (RESULT_re(I)); SPACE(2);
      FPRINT (RESULT_im(I)); SPACE(2);
      FPRINT (MAG(RESULT_re(I),RESULT_im(I)));
      I = I + 1;
   UNTIL I>(N/2-1) ;
   CRLF ;
END;
ENDPROC ;

PROCEDURE MAIN:
INTEGER N;
BEGIN
   PRINT "#Number of data points? ";
   N=FIX(FINPUT);
   CRLF;
   BEGIN
/*  start of timing   */
      inputfunction(.DATA, N) ;
      fastfourier(.DATA,.RESULT_RE,.RESULT_IM,N) ;
/*  end of timing   */
      outputresult(.DATA,.RESULT_RE,.RESULT_IM,N) ;
   END;
END;
```

The output results of the above program:

Number of data points? 64

```
0     1  0.125  0  0.125
1     1  0.118801  -0.0176803  0.120109
2     1  0.101647  -0.0312675  0.106348
3     1  0.077432  -0.0379243  0.0862204
4     1  0.0513167  -0.0368811  0.0631951
5     1  0.0282371  -0.0295286  0.0408567
6     1  0.0115892  -0.0187922  0.0220784
7     1  2.52019E-3  -8.0477E-3  8.43308E-3
8     0  0  0  0
9     0  1.56468E-3  4.07286E-3  4.36307E-3
10    0  4.39468E-3  4.47144E-3  6.26953E-3
11    0  6.30755E-3  2.63502E-3  6.83582E-3
12    0  6.3278E-3  3.48352E-4  6.33738E-3
13    0  4.69793E-3  -1.05139E-3  4.81414E-3
14    0  2.42651E-3  -1.15037E-3  2.68539E-3
15    0  6.34222E-4  -4.59947E-4  7.83446E-4
16    0  0  0  0
17    0  5.20991E-4  -6.89717E-4  8.64373E-4
18    0  1.63429E-3  -2.8324E-3  3.27008E-3
19    0  2.58408E-3  -5.91706E-3  6.4567E-3
20    0  2.82512E-3  -8.80455E-3  9.2467E-3
21    0  2.26601E-3  -0.0101962  0.010445
22    0  1.25557E-3  -9.16946E-3  9.25502E-3
23    0  3.50012E-4  -5.55294E-3  5.56396E-3
24    0  0  0  0
25    0  3.22647E-4  6.24422E-3  6.25255E-3
26    0  1.06642E-3  0.0117611  0.0118093
27    0  1.7717E-3  0.0153826  0.0154843
28    0  2.0304E-3  0.016466  0.0165907
29    0  1.70378E-3  0.0149524  0.0150492
30    0  9.8604E-4  0.0112447  0.0112878
31    0  2.86713E-4  6.00767E-3  6.01451E-3
```

The FFT in assembler assumes that the PAUSE line from the AM9511 is connected to NRDY from the 6809. If not, you must replace the original AMD macro to the following:

```
AMD   MACRO
      LDA   #1
      STA   APU+1
      LDA   APU+1
      BMI   *-2
      ENDM
```

This version is sometimes slower! It uses 5.43 seconds for 1024 points.
The assembler program uses 937 bytes of memory. The code is full relocatible (a must for PL/9). There must be at least 78 bytes free on stack plus memory for two temporary arrays of N*4 byte each. The required temporary memory on stack is also 78+8N, for 1024 points 8270 bytes.
The original sampled data points are stored in a byte array because the fastest A/D converter are 8-bit with flash mode. You can change the array to integer (16 bit) data. The original input data points are unchanged.
The FFT result coefficients in the arrays RES_RE and RES_IM are

in AMD floating-point format with is not compatible with the PL9 floating-point format. The procedure AMD_TO_PL9 converts one array of size N numbers to the PL9 format.
The assembling of the FFT program MUST be done with the "OPT EXP" for macro expansion and without the option "+C" on the calling line for generating all bytes. The XBASIC program CENPL9.BAS converts the output of the assembler into pruper GEN statements for PL9.

Here are now the FFT in assembler and the AMD to PL9 conversation program:

```
 TTL FFT: Uhrich Algorithm
 OPT PAG,EXP
 SPC 3
* *ASMPROC FASTFOURIER(INTNUMBER,INTNUMBR,INTNUMBR,INTNUMBR): INTEGER;
*
*   This routine is a assembler procedure for the PL9 compiler.
*   The "FASTFOURIER" procedure calculates the real and imaginary
*   cotficients using the Uhrich FFT algorithm.
*
*
*   Procedure call from PL9:
*
*   STACK_DEEP = FASTFOURIER(.DATA, .RES_RE, .RES_IM, N);
*
*   where STACK_DEEP   most deep stack value encountered
*                      in the procedure,
*         .DATA        pointer to the input integer data.
*                      of size N.
*                      This version processes Byte data.
*         .RES_RE      Pointer to real result Array. The
*                      size must be N*4. The format is in
*                      AMD floating point.
*         .RES_IM      Pointer to imaginary result Array.
*                      Same size and format as _RE.
*          N           Number of input data points.
*
*
****ZY
 SPC 3
*   HARDWARE DEPENDED ADDRESSES

APU EQU $F07E AMD9511 ADDRESS
 SPC 3
******************
*                *
*     MACROS     *
*                *
******************
 SPC 3
*   PUSH FP DATA ONTO APU STACK
*
*      FPUSH <r>     3,<r> TO 0,<r> => APU
*      FPUSH var     var+3,U TO var,U => PAU
*
FPUSH MACRO
 IFC &1,X,2
 IFC &1,Y,1
 IFNC &1,U,7
 LDD 2,&1
 STB APU
 STA APU
 LDD ,&1
 STB APU
 STA APU
 EXITM
 IFC &1,X++,2
 IFC &1,Y++,1
 IFNC &1,U++,2
 ERR &1 IN FPUSH NOT ALLOWED
 EXITM
 LDD &1+2,U
 STB APU
 STA APU
 LDD &1,U
 STB APU
 STA APU
 ENDM
 PAG
*   PULL FP DATA FROM APU STACK
*
*      FPULL <r>     APU => 0,<r> TO 3,<r>
*      FPULL <r++>   APU => r++ r++
*      FPULL var     APU => var,U TO var+3,U
*
FPULL MACRO
 IFC &1,X,2
 IFC &1,Y,1
 IFNC &1,U,7
 LDA APU
 LDB APU
 STD ,&1
 LDA APU
 LDB APU
 STD 2,&1
 EXITM
 IFC &1,X++,2
 IFC &1,Y++,1
 IFNC &1,U++,7
 LDA APU
 LDB APU
 STD ,&1
 LDA APU
 LDB APU
 STD ,&1
 EXITM
 LDA APU
 LDB APU
 STD &1,U
 LDA APU
 LDB APU
 STD &1+2,U
 ENDM
 PAG
* COMPUTE ARRAY INDEX
*
*       INDEX <array>,<index>
*
INDEX MACRO
 LDX &1,U GET ARRAY START ADDRESS
 LDD _&2,U GET INDEX VALUE
 LEAX D,X COMPUTE RESULT ADDRESS
 ENDM

*   ZERO ACCD
*
CLRD MACRO
 CLRA
 CLRB
 ENDM

*   EXECUTE AMD9511 COMMAND
*
*       AMD <operation>
*
AMD MACRO
 LDA _&1 GET COMMAND CODE
 STA APU+1 ISSUE COMMAND
 ENDM

*   PUSH INTEGER DATA ONTO APU STACK
*
IPUSH MACRO
 LDD &1,U
 STB APU
 STA APU
 ENDM

*   PULL INTEGER DATA FROM APU STACK
*
IPULL MACRO
 LDA APU
 LDB APU
 STD &1,U
 ENDM
 SPC 3
*   AMD9511 OPERATION CODES

_SIN EQU $82 FP SINUS
_COS EQU $83 FP COSINUS
_FADD EQU $90 FP ADD
_FSUB EQU $91 FP SUBTRACT
_FMUL EQU $92 FP MULTIPLY
_FDIV EQU $93 FP DIVIDE
_FNEG EQU $95 CHANGE FP SIGN
_FROT EQU $98 FP ROTATE STACK UP
_FEXC EQU $99 FP EXCHANGE TOS<=> NOS
_FUPI EQU $9A PUSH PI ON TOS
_ITOP EQU $9D INTEGER TO FP
```

```
_DUP EQU $B7 DUPLICATE PP TOS ONTO NOS
_IMUL EQU $EE INTEGER MULTIPLY
_IDIV EQU $EP INTEGER DIVIDE
 PAG
*   ARGUMENTS SAVED ON LOCAL STACK

RES_RE RMB 2 ...TO REAL RESULT
DATA RMB 2 POINTER TO INPUT DATA
N RMB 2 NUMBER OF DATA BYTES
RES_IM RMB 2 ...TO IMAGINARY RESULT
 SPC 3
*   VARIABLES ON LOCAL STACK
TMP_RE RMB 2
TMP_IM RMB 2
_AA RMB 2 INDEX
_BB RMB 2 INDEX
D RMB 2
_E RMB 2 INDEX
F RMB 2
J RMB 2
I RMB 2
R RMB 2
_R RMB 2 INDEX
IN2J RMB 2
N2 RMB 2
PNP2J RMB 2
IN2JN2 RMB 2
NSTAGE RMB 2
NP2J RMB 2
NP2JM1 RMB 2 NP2J-1
NP2J1 RMB 2
P2J2 RMB 2
P2J2M1 RMB 2 P2J2-1
N_RE RMB 4
M_IM RMB 4
W RMB 4
SINW RMB 4
COSW RMB 4
FN RMB 4

STKSIZ EQU * REQUIRED STACK SIZE
 PAG
 ORG $0000 PROGRAM IS RELOCATIBLE

FFT BRA FFTO SKIP OVER
VN FCB 1 VERSION NUMBER
 SPC 3
*   CONSTANTS: POWERS OF TWO

P2 FDB 1,2,4,8,16,32,64,128,256
 FDB 512,1024,2048,4096
 SPC 3
*   START HERE, ALLOCATE STACK SPACE

FFTO PSHS CC,DP,U,Y SAVE REGISTERS
 LEAU 8,S POINT TO ARGUMENTS
 LEAS -STKSIZ+8,S MAKE ROOM ON STACK
 PULU D,X GET N AND RES_IM
 PSHS D,X SAVE IT
 PULU X,Y GET RES_RE AND DATA PNTR
 PSHS X,Y SAVE IT
 TFR S,U SET FRAME POINTER
 ASLD SIZE FOR FP ARRAYS
 ASLD ...TIMES FOUR
 COMA MAKE COMPLEMENT
 COMB ...AND A
 ADDD 1 ...NEGATIVE VALUE
 LEAS D,S SPACE FOR TMP_RE
 STS TMP_RE,U SAVE ADDRESS
 LEAS D,S SPACE FOR TMP_IM
 STS TMP_IM,U SAVE ADDRESS
 SEI (LINE CAN BE DELETE)
 LDA OPU>>8 GET MSB APU ADDRESS
 TFR A,DP SETUP DP REGISTER
 STDP APU>>8 INFORM ASSEMBLER
*   SOME SETUPS
 LDD N,U
 LDX -1
NSTG LEAX 1,X
 LSRD
 BNE NSTG
 TSTA
 BNE NSTG
 STX NSTAGE,U
 LDD N,U
 LSRA
 RORB
 STD N2,U
 STB APU
 STA APU
 AMD ITOF FLOAT(N/2)
 AMD PUP1 B=FLOAT(N), A=PI
 AMD PEXG B=PI, A=FLOAT(N/2)
 AMD FDIV B/A=>A
 FPULL W W=PI/FLOAT(N/2)
 IPUSH N
 AMD ITOF
 FPULL FN FN=FLOAT((N)

*  CONVERT INTEGER DATA TO REAL
 LDX TMP_RE,U
 LDY DATA,U
 LDD N,U
 PSHS D,U
 LDU TMP_IM,U
DATREAL CLRA FOR 16-BIT INTEGER DATA,
 LDB ,Y+ ...CHANGE TO "LDD ,Y++"
 STB APU
 STA APU
 AMD ITOF
 CLRD
 STD ,U++
 STD ,U++ ZERO TEMP_IM
 FPULL X++ SETUP TEMP_RE
 LDD ,S
 SUBD 1
 STD ,S
 BNE DATREAL
 PULS D,U
 SPC 3
*    j = 1 ;
*    REPEAT
 LDD 1
L2 STD J,U

*       NP2J  = N/P2(j) ;
 ASLD INDEX STILL IN ACCD
 LEAX P2,PCR
 LEAX D,X X=> P2(J)
 LDD N,U
 STB APU
 STA APU
 LDD ,X
 STB APU
 STA APU
 AMD IDIV
 LDA APU
 LDB APU
 STD NP2J,U
 SUBD 1
 STD NP2JM1,U
*       NP2J1 = N/P2(J-1);
 LDD N,U
 STB APU
 STA APU
 LDD -2,X AT P2(J-1)
 STB APU
 STA APU
 AMD IDIV

*       P2J2 = P2(J)/2 ;
 LDD ,X P2(J)
 LSRD /2
 STD P2J2,U
 SUBD 1
 STD P2J2M1,U COUNTER END
 LDA APU
 LDB APU
 STD NP2J1,U GET PREVIOUS RESULT
 SPC 3
*       i = 0 ;
```

```
*       REPEAT
 CLRD
L3 STD I,U

*          in2j    = i*NP2J ;
 LEAX NP2J,U
 STB APU
 STA APU
 LDD ,X
 STB APU
 STA APU
 AMD IMUL
 LDA APU
 LDB APU
 STD IN2J,U

*          arg     = W*FLOAT(in2j) ;
*          cosw    = COS(arg) ;
*          sinw    = SIN(arg) ;
 STB APU
 STA APU IN2J IS STILL IN ACCD
 AMD ITOF
 FPUSH W
 AMD FMUL
 AMD DUP DUPLICATE "arg"
 AMD COS
 FPULL COSW
 AMD SIN
 FPULL SINW

*          F       = i*NP2J1 ;
*          FNP2J   = F + NP2J;
*          in2jN2  = in2j + N2;
 LDD I,U
 LEAX NP2J1,U
 STB APU
 STA APU
 LDD ,X
 STB APU
 STA APU
 AMD IMUL
 LDD IN2J,U
 ADDD N2,U
 STD IN2JN2,U
 LDA APU
 LDB APU
 STD F,U PREVIOUS RESULT
 ADDD NP2J,U
 STD FNP2J,U
 PAG
*                    INNER LOOP
*                    ----------

*          r = 0 ;
*          REPEAT
 LDX 0
L4 STX R,U

*             A          = r + in2j ;
*             B          = r + F ;
*             C          = r + FNP2J ;
*             E          = r + in2jN2 ;
 LDD R,U
 ADDD FNP2J,U
 ASLD
 ASLD

*             M_re       =  TEMP_re(C)*cosw ;
*             M_im       = -TEMP_re(C)*sinw ;
 LDX TMP_RE,U
 LEAX D,X
 FPUSH X
 AMD DUP DUPLICATE "TEMP_re(C)"
 FPUSH COSW
 AMD FMUL
 FPULL M_RE
 AMD FNEG NEGATE TEMP_RE(C)
 FPUSH SINW
 AMD FMUL
 LDD R,U
 ADDD F,U
 ASLD
 ASLD
 STD _BB,U "B" AS INDEX
 LDX TMP_RE,U
 LEAX D,X INDEX STILL IN ACCD
 FPULL M_IM

*             RESULT_re(A) = TEMP_re(B) + M_re ;
*             RESULT_re(E) = TEMP_re(B) - M_re ;
 FPUSH M_RE
 AMD DUP DUPLICATE M_RE
 FPUSH X
 AMD DUP DUPLICATE "TEMP.re(B)"
 AMD FROT ROTATE UP
 AMD FADD
 LDD R,U
 ADDD IN2J,U
 ASLD
 ASLD
 STD _AA,U "A" AS INDEX
 INDEX RES_RE,AA
 FPULL X
 AMD FSUB
 LDD R,U
 ADDD IN2JN2,U
 ASLD
 ASLD
 STD _E,U "E" AS INDEX
 INDEX RES_RE,E
 FPULL X

*             RESULT_im(A) = TEMP_im(B) + M_im ;
*             RESULT_im(E) = TEMP_im(B) - M_im ;
 INDEX TMP_IM,BB
 FPUSH M_IM
 AMD DUP DUPLICATE M_IM
 FPUSH X
 AMD DUP DUPLICATE "TEMP_im(B)"
 AMD PROT ROTATE STACK UP
 AMD FADD
 INDEX RES_IM,AA
 FPULL X
 AMD FSUB
 INDEX RES_IM,E
 FPULL X

*             r = r + 1 ;
*          UNTIL r>(NP2J-1) ;
 LDX R,U
 LEAX 1,X
 CMPX NP2JM1,U
 LBLS L4
 PAG
*          i = i + 1 ;
*       UNTIL i>(P2J2-1) ;
 LDD I,U
 ADDD 1
 CMPD P2J2M1,U
 LBLS L3
*       r = 0 ;
*       REPEAT
*          TEMP_re(r) = RESULT_re(r) ;
*          TEMP_im(r) = RESULT_im(r) ;
*          r = r + 1 ;
*       UNTIL r>(N-1) ;
 LDX N,U
 LDY TMP_RE,U
 PSHS U
 LDU RES_RE,U
LL1 LDD ,U++
 STD ,Y++
 LDD ,U++
 STD ,Y++
 LEAX -1,X
 BNE LL1
 LDU ,S
 LDX N,U
 LDY TMP_IM,U
 LDU RES_IM,U
```

```
LL2 LDD .U++
 STD ,Y++
 LDD ,U++
 STD ,Y++
 LEAX -1,X
 BNE LL2
 PULS U

*       j = j + 1 ;
*    UNTIL j>NSTAGE ;
 LDD J,U
 ADDD 1
 CMPD NSTAGE,U
 LBLS L2

*    r = 0 ;
*    REPEAT
*       RESULT_re(r) = RESULT_re(r)/FN ;
*       RESULT_im(r) = RESULT_im(r)/FN ;
*       r = r + 1;
*    UNTIL r>(N-1) ;
 LDX N,U
 LDY RES_RE,U
 FPUSH FN
RR1 AMD DUP
 FPUSH Y
 AMD FEXC
 AMD FDIV
 FPULL Y++
 LEAX -1,X
 BNE RR1
 LDX N,U
 LDY RES_IM,U
RR2 AMD DUP
 FPUSH Y
 AMD FEXC
 AMD FDIV
 FPULL Y++
 LEAX -1,X
 BNE RR2

*    END ;
 TFR S,U SAVE STACK POINTER
 SUBD 4 RESULT IS STACK DEEP
 TFR U,S
 LEAS STKSIZ,S

* ENDPROC ;
 PULS CC,DP,U,Y,PC

 END



* +ASMPROC AMD_TO_PL9(INTEGER,INTEGER);
*        AMD_TO_PL9(INTEGER .BUFFER,COUNT);
*
*   CONVERT ARRAY POINTED BY .BUFFER WITH LENGTH COUNT
*   FROM AM9511 FLOATING-POINT FORMAT TO PL9 FLOATING-
*   POINT FORMAT.

AMDPL9  PSHS  U             SAVE REGISTER
        LDX   4,S           GET COUNT
        LDU   6,S           ...AND DATA POINTER
CONVERT LSR   1,U           MAKE ROOM FOR PL9 SIGN
        ROR   2,U           SHIFT ONE BIT
        ROR   3,U           ...TO RIGHT
        LDA   ,U            GET EXPONENT
        BPL   POS           SKIP IF POSITIVE RESULT
        COM   1,U           ..RESULT IS NEGATIVE
        COM   2,U           ...AND MANTISSA MUST
        COM   3,U           ...FOR PL9
        INC   3,U           ...Twos-complement
        BNE   POS           ...NUMBER
        INC   2,U           ...SO NEGATE THE
        BNE   POS           ...MANTISSA FROM
        INC   1,U           ..AMD9511
POS     ANDA  $7F           MASK MANTISSA
        BITA  $40           EXPONENT POSITIVE ?
        BEQ   *+4           SKIP IF SO
        ORA   $80           ELSE MAKE NEGATIVE
        STA   ,U            SAVE BACK
        LEAU  4,U           NEXT FP NUMBER
        LEAX  -1,X          LAST NUMBER ?
        BNE   CONVERT       LOOP IF NOT
        PULS  U             RECOVER REGISTER
        RTS                 BACK TO PL9
```

Here are now the complete PL9 program with FFT in assembler:

```
/*   FFT: The Uhrich Algorithm in PL9   */

GLOBAL BYTE DATA(1024):       /* Input data from A/D converter */
       REAL RESULT_re(1024),  /* Floating-Point result: real   */
            RESULT_im(1024);  /*                  and imaginary part   */

INTEGER P2 1,2,4,8,16,32,64,128,256,512,1024,2048;     /* 2^X  */


INCLUDE TRUFALSE.DEF ;
INCLUDE IOSUBS.LIB ;
INCLUDE SCIPACK.LIB ;
INCLUDE REALCON.LIB ;
INCLUDE REALIO.LIB ;


PROCEDURE inputfunction(BYTE .DATA: INTEGER N): INTEGER i;
/*A 12.514 pulse is defined in this function*/
BEGIN
   i = 0 ;
   REPEAT
      DATA(i) = i ;
      i = i + 1 ;
   UNTIL i > N/8 ;
   i = N/8 ;
   REPEAT
      DATA(i) = 0 ;
      i = i + 1 ;
   UNTIL i > (N-i) ;
END;
ENDPROC ;

ASMPROC FASTFOURIER(INTEGER,INTEGER,INTEGER,INTEGER): INTEGER;
   GEN $20,$1B,$01,$00,$01,$00,$02,$00,$04,$00,$08,$00,$10,$00,$20;
   GEN $40,$80,$00,$01,$00,$02,$00,$04,$00,$08,$00,$10,$00,$14;
   GEN $69,$33,$68,$32,$68,$86,$37,$16,$34,$16,$37,$30,$34,$30,$1F;
   GEN $43,$58,$49,$58,$49,$43,$53,$C3,$00,$01,$32,$EB,$10,$EF,$48;
   GEN $32,$EB,$10,$EF,$4A,$1A,$10,$86,$F0,$1F,$8B,$EC,$44,$8E,$FF;
   GEN $FF,$30,$01,$44,$56,$26,$FA,$40,$26,$F7,$AF,$C8,$26,$EC,$44;
   GEN $44,$56,$ED,$C8,$20,$07,$7E,$97,$7E,$86,$90,$97,$7F,$86,$9A;
   GEN $97,$7F,$86,$99,$97,$7F,$86,$93,$97,$7F,$96,$7E,$06,$7E,$ED;
   GEN $C8,$3A,$96,$7E,$06,$7E,$ED,$C8,$3C,$EC,$44,$07,$7E,$97,$7E;
   GEN $86,$9D,$97,$7F,$96,$7E,$06,$7E,$ED,$C8,$46,$96,$7E,$06,$7E;
   GEN $ED,$C8,$48,$AE,$48,$10,$AE,$42,$EC,$44,$34,$46,$EE,$4A,$4F;
   GEN $E6,$A0,$D7,$7E,$97,$7E,$86,$90,$97,$7F,$4F,$5F,$ED,$C1,$ED;
   GEN $C1,$96,$7E,$06,$7E,$ED,$81,$96,$7E,$06,$7E,$ED,$81,$EC,$E4;
   GEN $83,$00,$01,$ED,$E4,$26,$DA,$35,$46,$CC,$00,$01,$ED,$C8,$16;
   GEN $58,$49,$30,$8D,$FF,$28,$30,$86,$EC,$44,$D7,$7E,$97,$7E,$EC;
   GEN $84,$D7,$7E,$97,$7E,$86,$EF,$97,$7F,$96,$7E,$06,$7E,$ED,$C8;
   GEN $2B,$83,$00,$01,$ED,$C8,$2A,$EC,$44,$D7,$7E,$97,$7E,$EC,$1E;
   GEN $D7,$7E,$97,$7E,$86,$EF,$97,$7F,$EC,$84,$44,$56,$ED,$C8,$2E;
   GEN $83,$00,$01,$ED,$C8,$30,$96,$7E,$06,$7E,$ED,$C8,$2C,$4F,$5F;
   GEN $ED,$C8,$18,$30,$C8,$28,$D7,$7E,$97,$7E,$EC,$84,$D7,$7E,$97;
   GEN $7E,$86,$EE,$97,$7F,$96,$7E,$06,$7E,$ED,$C8,$1E,$D7,$7E,$97;
   GEN $7E,$86,$9D,$97,$7F,$EC,$C8,$3C,$D7,$7E,$97,$7E,$EC,$C8,$3A;
   GEN $D7,$7E,$97,$7E,$86,$92,$97,$7F,$86,$87,$97,$7F,$86,$83,$97;
   GEN $7F,$96,$7E,$06,$7E,$ED,$C8,$42,$96,$7E,$06,$7E,$ED,$C8,$44;
   GEN $86,$82,$97,$7F,$96,$7E,$06,$7E,$ED,$C8,$3E,$96,$7E,$06,$7E;
   GEN $ED,$C8,$40,$EC,$C8,$18,$30,$C8,$2C,$D7,$7E,$97,$7E,$EC,$84;
   GEN $D7,$7E,$97,$7E,$86,$EE,$97,$7F,$EC,$C8,$1E,$E3,$C8,$2D,$ED;
   GEN $C8,$24,$96,$7E,$06,$7E,$ED,$C8,$14,$E3,$C8,$28,$ED,$C8,$22;
   GEN $8E,$00,$00,$AF,$C8,$1A,$EC,$C8,$1A,$E3,$C8,$22,$58,$49,$58;
   GEN $49,$AE,$48,$30,$88,$EC,$02,$D7,$7E,$97,$7E,$EC,$84,$D7,$7E;
   GEN $97,$7E,$86,$87,$97,$7F,$EC,$C8,$44,$D7,$7E,$97,$7E,$EC,$C8;
   GEN $42,$D7,$7E,$97,$7E,$86,$92,$97,$7F,$96,$7E,$06,$7E,$ED,$C8;
   GEN $32,$96,$7E,$06,$7E,$ED,$C8,$34,$86,$95,$97,$7F,$EC,$C8,$40;
   GEN $D7,$7E,$97,$7E,$EC,$C8,$3E,$D7,$7E,$97,$7E,$86,$92,$97,$7F;
   GEN $8C,$C8,$1A,$E3,$C8,$14,$58,$49,$58,$49,$ED,$4E,$AE,$48,$30;
   GEN $8B,$96,$7E,$06,$7E,$ED,$C8,$3b,$96,$7E,$06,$7E,$ED,$C8,$38;
   GEN $EC,$C8,$34,$D7,$7E,$97,$7E,$8C,$C8,$32,$D7,$7E,$97,$7E,$86;
   GEN $87,$97,$7F,$EC,$02,$D7,$7E,$97,$7E,$EC,$84,$D7,$7E,$97,$7E;
   GEN $86,$87,$97,$7F,$86,$98,$97,$7F,$86,$90,$97,$7F,$EC,$C8,$1A;
   GEN $E3,$C8,$1E,$58,$49,$58,$49,$ED,$4C,$AE,$C4,$EC,$4C,$30,$88;
   GEN $96,$7E,$06,$7E,$ED,$84,$96,$7E,$06,$7E,$ED,$02,$86,$91,$97;
   GEN $7F,$EC,$C8,$1A,$E3,$C8,$24,$58,$49,$58,$49,$ED,$C8,$12,$AE;
   GEN $C4,$EC,$C8,$12,$30,$8B,$96,$7E,$06,$7E,$ED,$84,$96,$7E,$06;
   GEN $7E,$ED,$02,$AE,$4A,$EC,$4E,$30,$8B,$EC,$C8,$38,$D7,$7E,$97;
   GEN $7E,$EC,$C8,$36,$D7,$7E,$97,$7E,$86,$87,$97,$7F,$EC,$02,$D7;
   GEN $7E,$97,$7E,$EC,$84,$D7,$7E,$97,$7E,$86,$87,$97,$7F,$86,$98;
   GEN $97,$7F,$86,$90,$97,$7F,$AE,$4E,$EC,$4C,$3D,$8B,$96,$7E,$06;
   GEN $7E,$ED,$84,$96,$7E,$06,$7E,$ED,$02,$86,$91,$97,$7F,$AE,$46;
   GEN $EC,$C8,$12,$30,$8B,$96,$7E,$06,$7E,$ED,$84,$96,$7E,$06,$7E;
   GEN $ED,$02,$AE,$C8,$1A,$3D,$01,$AC,$C8,$2A,$10,$23,$FE,$BA,$EC;
   GEN $C8,$18,$C3,$00,$01,$10,$A3,$C8,$30,$10,$23,$FE,$22,$AE,$44;
   GEN $10,$AE,$48,$34,$40,$EE,$C4,$EC,$C1,$ED,$A1,$EC,$C1,$ED,$A1;
   GEN $30,$1F,$26,$F4,$EE,$E4,$AE,$44,$10,$AE,$4A,$EE,$46,$EC,$C1;
   GEN $ED,$A1,$EC,$C1,$ED,$A1,$3D,$1F,$26,$F4,$35,$4D,$EC,$C8,$16;
   GEN $C1,$00,$01,$10,$A3,$CH,$26,$10,$23,$FD,$9A,$AE,$44,$10,$AE;
```

```
GEN $C4,$EC,$C8,$48,$D7,$7E,$97,$7E,$EC,$CA,$46,$D7,$7E,$97,$7E;
GEN $86,$B7,$97,$7F,$EC,$22,$D7,$7E,$97,$7E,$EC,$A4,$D7,$7E,$97;
GEN $7E,$86,$99,$97,$7F,$86,$93,$97,$7F,$96,$7E,$D6,$7E,$ED,$A1;
GEN $96,$7E,$D6,$7E,$ED,$A1,$30,$1F,$26,$D8,$AE,$44,$1D,$AE,$46;
GEN $86,$B7,$97,$7F,$EC,$22,$D7,$7E,$97,$7E,$EC,$A4,$D7,$7E,$97;
GEN $7E,$86,$99,$97,$7F,$86,$93,$97,$7F,$96,$7E,$D6,$7E,$ED,$A1;
GEN $96,$7E,$D6,$7E,$ED,$A1,$10,$1F,$26,$D8,$1F,$40,$83,$00,$04;
GEN $1F,$34,$32,$E8,$4A,$35,$89;

ASMPROC AND TO PL9(INTEGER,INTEGER);
   GEN $34,$40,$AE,$64,$EE,$66,$64,$41,$66,$42,$66,$43,$A6,$C4,$2A;
   GEN $10,$61,$41,$61,$42,$01,$41,$60,$43,$26,$06,$66,$42,$26,$08;
   GEN $6C,$41,$84,$7E,$85,$40,$27,$02,$8A,$80,$A7,$C4,$33,$44,$30;
   GEN $1F,$26,$06,$05,$40,$39;



PROCEDURE MAG(REAL x,y) ;
ENDPROC REAL SQR(x*x+y*y) ;


PROCEDURE outputresult(BYTE    .DATA:       /* Input data          */
                       REAL    .RESULT_RE,  /* Real result         */
                               .RESULT_IM:  /* Imaginary result    */
                       INTEGER N):          /* Number of points    */
INTEGER i ;
BEGIN
   i = 0;
   REPEAT
      CRLF ;
      PRINTINT(i);            SPACE(4);
      FPRINT (DATA(i));       SPACE(2);
      FPRINT (RESULT_re(i)); SPACE(2);
      FPRINT (RESULT_im(i)); SPACE(2);
      FPRINT (MAG(RESULT_re(i),RESULT_im(i)));
      i = i + 1;
   UNTIL i>(N/2-1) ;
   CRLF ;
END;
ENDPROC ;

PROCEDURE MAIN:
INTEGER N;
BEGIN
   PRINT "NNumber of data points? ";
   N=FIX(FINPUT);
   CRLF;
   BEGIN
/*   timing starts here    */
      inputfunction(.DATA, N) ;
      fastfourier(.DATA,.RESULT_RE,.RESULT_IM,N) ;
      amp_to_pl9(.RESULT_RE,N);
      amp_to_pl9(.RESULT_IM,N);
/*   timing ends here      */
      outputresult(.DATA,.RESULT_RE,.RESULT_IM,N) ;
   END;
END;
```

The output results of the above program:

```
Number of data points? 64

0    |  0.125   0   0.125
1    |  0.118801  -0.0176801  0.120109
2    |  0.101647  -0.0312074  0.106348
3    |  0.0774319  -0.0379243  0.0862204
4    |  0.0513167  -0.0368811  0.0631951
5    |  0.028247  -0.0295286  0.0408567
6    |  0.0115892  -0.0187922  0.0220784
7    |  2.520198E-3  -8.04778E-3  8.433088E-3
8    |  0  0  0  0
9    |  0  1.964688E-3  4.072868E-4  4.96307E-3
10   |  0  4.39468E-3  4.471448E-3  6.26953E-3
11   |  0  6.307568E-3  2.635018E-3  6.83583E-3
12   |  0  6.32779E-3  3.48357E-4  6.33737E-3
13   |  0  4.69794E-3  -1.051398E-3  4.81415E-3
14   |  0  2.42652E-3  -1.15037E-3  2.68539E-3
15   |  0  6.34224E-4  -4.59939E-4  7.83443E-4
16   |  0  0  0  0
17   |  0  5.20989E-4  -6.89717E-4  8.64372E-4
18   |  0  1.63429E-3  -2.81241E-3  3.27009E-3
19   |  0  2.584088E-3  -5.91707E-3  6.456672E-3
20   |  0  2.825126E-3  -8.80455E-3  9.24667E-3
21   |  0  2.266018E-3  -0.0111962  0.010445
22   |  0  1.25557E-3  -9.169458E-3  9.255028E-3
23   |  0  3.500128E-4  -5.552948E-3  5.56396E-3
24   |  0  0  0  0
25   |  0  3.22648E-4  6.24422E-3  6.25255E-3
26   |  0  1.06643E-3  0.0117611  0.0118093
27   |  0  1.7717E-3  0.0153826  0.0154843
28   |  0  2.0304E-3  0.016466  0.0165907
29   |  0  1.7047NE-3  0.0149525  0.0150492
30   |  0  9.8604E-4  0.0112447  0.0112878
31   |  0  2.8671E-4  6.007665E-3  6.0145E-3
```

The GENPL9.BAS BASIC program:

```
10 REM*************************************************
20 REM                                                 *
30 REM            G E N P L 9 . B A S                  *
40 REM                                                 *
50 REM     Generates GEN statements for PL9,           *
60 REM           from an assembly file                 *
70 REM   Note: the assembly must be with OPT EXP       *
80 REM         and calling line must not be =G         *
90 REM           Version from 7.Apr. 86                *
100 REM                                                *
110 REM*************************************************
120 DIM A17(10)
130 REM
140 GOSUB 890 : REM*** INITIALIZE PARAMETERS
150 REM
160 REM
170 REM ********** M A I N   L O O P **********
180 REM
190 INPUT LINE 1,A$ : IF A$="" THEN 190
200 IF INSTR(1,A$,L$)<>0 THEN GOTO 380
210 IF INSTR(1,A$,T$)<>0 THEN 190
220 P17=INSTR(1,A$," =");REM*** CHECK FOR PROCEDURE HEADER
230 IF P17<>0 THEN PRINT 1518,RIGHT$(A$,LEN(A$)-P17-2) : L17=L17+1 : GOTO 180
240 A$=MID$(A$,A17(1),12)          :REM*** OP-CODES
250 M$=RIGHT$(A$,LEN(A$)-A17(2))   :REM*** MNEMONICS
260 D17=0 : X17=1
270 IF MID$(A$,1,1)=" " THEN GOTO 360:REM*** EQU, SET, COMMENTS...
280 REM
290 REM*** SCANN INPUT LINE
300 IF X17>LEN(A$) THEN 350
310 IF MID$(A$,X17,1)=" " THEN X17=X17+1 : GOTO 300
320 B$=B$+U$+MID$(A$,X17,J)+"," : D17=1
330 Y17=Y17-1 : B17=B17+1 : X17=X17+2
340 IF Y17=0 THEN GOSUB 520   :REM*** PRINT LINE IF FULL
350 GOTO 300
360 GOSUB 520 : REM*** PRINT OUT GENERATED LINE
370 GOTO 180
380 REM
390 REM
400 REM   **********************
410 REM   *   PROGRAM END       *
420 REM   **********************
430 REM
440 IF C17=0 THEN 460 :REM*** CHECK FOR NON-PRINTED CODE
450 Y17=0:IF LEN(B$)>LEN(C$) THEN D17=1:GOSUB 520
460 PRINT:PRINT:PRINT
470 PRINT "Total";L17;"generated lines with";B17;"bytes"
480 IF U17<>0 THEN PRINT "Result stands on file ";F2$
490 PRINT "   Generation complete."
500 GOTO 1060
510 IF O17<>0 THEN CLOSE O17
520 REM
530 REM
540 REM   **********************
550 REM   *   PRINT OUT LINE   *
560 REM   **********************
570 REM
580 IF C17=0 THEN GOTO 600 :REM*** SKIP IF NON-COMPRESSED
590 IF Y17<>0 THEN GOTO 690
600 IF RIGHT$(B$,1)="," THEN B$=LEFT$(B$,LEN(B$)-1)+S$ :REM*** ";"
610 IF D18=0 AND M$="" THEN 680   :REM*** NOTHING TO PRINT
620 T18=A18(4)
630 IF D18<>0 THEN 640   :REM*** SKIP IF GENERATED OP-CODES
635 B$="":T18=3
636 IF LEFT$(M$,1)="=" THEN M$=RIGHT$(M$,LEN(M$)-1)
640 IF T18=A18(4) THEN M$=LEFT$(M$,A18(3))
650 IF D18<>0 THEN PRINT 1518,B$;
660 IF C18=0 THEN PRINT 1518,TAB(T18);S1$+M$;TAB(70);S9$;
670 IF D18<>0 OR C18=0 THEN PRINT 1518 : L18=L18+1
680 B$=G$ : Y18=15
690 RETURN
700 REM
710 REM
720 REM   **********************
730 REM   *   ERROR ROUTINE    *
740 REM   **********************
750 REM
760 IF ERR=8 THEN RESUME 380
770 IF ERR<>4 THEN ON ERROR GOTO

780 PRINT:PRINT CHR$(7);"FILE ";F1$; " NOT FOUND.":PRINT
790 RESUME 810
800 STOP
810 REM
820 REM
830 REM   **********************
840 REM   *   INITALIZATION    *
850 REM   **********************
860 REM
870 ON ERROR GOTO 710
880 S1$="/* " : S9$=" */" : U$="$"
890 S$=";" : G$="   GEN " : B$=G$
891 E$="ERROR(S) DETECTED" :REM*** END STRING
892 T$="TSC ASSEMBLER"     :REM*** PAGE STRING
900 Y18=15       :REM*** MAX BYTES PER LINE
910 O18=0        :REM*** OUTPUT CHANNEL
920 L18 =0       :REM*** GENERATED LINES
930 B18=0        :REM*** GENERATED BYTES
940 A18(1)=8     :REM*** ASMB: START COLUMN OF OP CODES
950 A18(2)=15    :REM*** ASMB: START COLUMN OF MNEMONIC
```

```
960 A1$(3)=4ll   :REM*** RIGHT COLUMN OF GENERATED "*/"
970 A1$(4)=27    :REM*** LEFT COLUMN OF GENERATED "/*"
980 INPUT "File to read ",F1$
990 OPEN 0,0 F1$ AS 1
1000 INPUT "Compressed format? (Y/N) ";C$
1010 IF C$="Y" THEN C18=1 ELSE C18=0
1020 INPUT "Output to T(erminal), P(rinter), or D(isk) ";O$
1030 IF O$="P" THEN OPEN "U.PRINT" AS 0
1040 IF O$<>"D" THEN 1070
1050 INPUT "File to write ";F2$
1060 O18=2 : OPEN NEW F2$ AS O18
1070 PRINT : PRINT : RETURN
1080 END
```

Here the Uhrich algorithm in TSC PASCAL. After this program the PL9 programm was been written:

```
{                    The Uhrich Algorithm                    }
{   This algorithm is an non-in-place type FFT. The Uhrich   }
{   algorithm was first published in June of 1969. This      }
{   method results in very compact programs (see Uhrich,     }
{   1969). This process does not require a bit reversing     }
{   procedure as in the Coolet-Tukey algorithm.              }

CONST
   N      = 64 ; {number of data points}
   NSTAGE = 6 ; {power of two of the number of data points}
   pi     = 3.141592654 ;

TYPE
   complex = RECORD
      re : REAL ;
      im : REAL
   END ;

VAR {a two dimensional array is used to store input data and results}
   x : ARRAY [1..2,0..N] OF complex ;



PROCEDURE inputfunction ;
{A 12.5Hz pulse is defined in this function}
VAR
   i : INTEGER ;
BEGIN
   FOR i := 0 TO (N DIV 8) DO
      BEGIN
         x[1,i].re := 1.0 ;
         x[1,i].im := 0.0
      END ;
   FOR i := (N DIV 8) TO N DO
      BEGIN
         x[1,i].re := 0.0 ;
         x[1,i].im := 0.0
      END
END ;


FUNCTION P2(x : INTEGER) : INTEGER ;
BEGIN
   CASE x OF
      0  : P2 :=    1 ;
      1  : P2 :=    2 ;
      2  : P2 :=    4 ;
      3  : P2 :=    8 ;
      4  : P2 :=   16 ;
      5  : P2 :=   32 ;
      6  : P2 :=   64 ;
      7  : P2 :=  128 ;
      8  : P2 :=  256 ;
      9  : P2 :=  512 ;
      10 : P2 := 1024 ;
      11 : P2 := 2048
   END
END ;


PROCEDURE fastfourier ; {Uhrich algorithm}
VAR
   A,B,C,D,E,F,j,i,r,in2j : INTEGER ;
   M                      : complex ;
   W,minw,conw            : REAL;
BEGIN
   FOR j := 1 TO NSTAGE DO
      BEGIN
         FOR i := 0 TO (P2(j) DIV 2)-1 DO
            BEGIN
               W     := 2.0*pi/CONV(N) ;
               in2j  := i*(N DIV P2(j)) ;
               conw  := COS(W*CONV(in2j)) ;
               minw  := SIN(W*CONV(in2j)) ;
               D     := i*(N DIV P2(j)) ;
               F     := i*(N DIV P2(j-1)) ;
               FOR r := 0 TO (N DIV P2(j))-1 DO
                  BEGIN
                     A          := r + D ;
                     B          := r + F ;
                     C          := r + F + (N DIV P2(j)) ;
                     E          := r + D + (N DIV 2) ;
                     M.re       := x[1,C].re*conw ;
                     M.im       := -x[1,C].re*minw ;
                     x[2,A].re := x[1,B].re + M.re ;
                     x[2,A].im := x[1,B].im + M.im ;
                     x[2,E].re := x[1,B].re - M.re ;
                     x[2,E].im := x[1,B].im - M.im
                  END
            END ;
         FOR r := 0 TO N-1 DO
            BEGIN
               x[1,r].re := x[2,r].re ;
               x[1,r].im := x[2,r].im
            END
      END ;
   FOR r := 0 TO N-1 DO
      BEGIN
         x[1,r].re := x[1,r].re/CONV(N) ;
         x[1,r].im := x[1,r].im/CONV(N)
      END
END ;


FUNCTION MAG(x,y : REAL) : REAL ;
BEGIN
   MAG := SQRT(SQR(x)+SQR(y))
END ;
PROCEDURE outputresult ;
VAR
   i : INTEGER ;
BEGIN
   FOR i := 0 TO (N DIV 2)-1 DO
      BEGIN
         WRITELN ;
         WRITE(i:3,'   ',x[1,i].re:16:11,'  ',x[1,i].im:16:13);
         WRITE('  ',MAG(x[1,i].re,x[1,i].im):16:13)
      END ;
   WRITELN
END ; {fastfouriertransform}


BEGIN {Fast Fourier Transform MAIN program block}
   inputfunction ;
   fastfourier ;
   outputresult
END . {fastfourierfunction}
```

The output results of the above program:

```
 0   0.125000000000000   0.000000000000000   0.125000000000000
 1   0.118800539251?   -0.017680260463?   0.120108949444?
 2   0.101647332923?   -0.031267455089?   0.106347703493?
 3   0.077431930514?   -0.037924337026?   0.086220411588?
 4   0.051316843014?   -0.036881066616?   0.063195056471?
 5   0.028237031096?   -0.029528625261?   0.040856696331?
 6   0.011589156264?   -0.018792170101?   0.022078364976?
 7   0.012520199173?   -0.008047696124?   0.008433076354?
 8   0.000000000000?   0.000000000000?   0.000000000000?
 9   0.001564678774?   0.004072862028?   0.004363075150?
10   0.004394676380?   0.004471445439?   0.006269529885?
11   0.006301755268?   0.002635011622?   0.006835827195?
12   0.006327790638?   0.000348362014?   0.006337372519?
13   0.004697930031?   -0.001051382125?   0.004814140728?
14   0.002426514489?   -0.001150368680?   0.002685390228?
15   0.000634221659?   -0.000459931955?   0.000783437627?
16   0.000000000000?   0.000000000000?   0.000000000000?
17   0.000520991018?   -0.000689718052?   0.000864374127?
18   0.001634290700?   -0.002832409274?   0.003270083851?
19   0.002584075429?   -0.005917062625?   0.006456707825?
20   0.002825122444?   -0.008804551077?   0.009246698681?
21   0.002266005385?   -0.010196219795?   0.010444983414?
22   0.001255568508?   -0.009169452386?   0.009255015361?
23   0.000350011880?   -0.005552937915?   0.005563957926?
24   0.000000000000?   0.000000000000?   0.000000000000?
25   0.000322647318?   0.006244218442?   0.006252548700?
26   0.001066425009?   0.011761105868?   0.011809355340?
27   0.001771696293?   0.015382609098?   0.015484300772?
28   0.002030402594?   0.016466020297?   0.016590731120?
29   0.001703778528?   0.014952453268?   0.015049209946?
30   0.000986039696?   0.011244678157?   0.011286782765?
31   0.000286718846?   0.006007667979?   0.006014505985?
```

That's all !

- - -

MOTOROLA INC.

Microprocessor Products Group
P.O. Box 3600
Austin, Texas 78764

EDITORIAL CONTACT:
Bob King
512/928-6141

INQUIRY RESPONSE:
R.Q. Green
P.O. Box 52073
Phoenix, AZ 85072

MC68HC11A8 SINGLE-CHIP MCU

REDUCED BELOW $20 IN LOW QUANTITIES

AUSTIN, TEXAS, MAY 13, 1986... The Microcomputer Operation of Motorola's Microprocessor Products Group has reduced the price of the most highly-integrated, single-chip microcomputer produced in any technology-- the MC68HC11A8-- to $19.95 for 1000 minimum order quantities. Increased automation and high volumes have combined to yield this more cost effective solution.

In addition, a version of the MC68HC11A8 is available that contains the BUFFALO Monitor Routine-- the MC68HC11A8FN1. The BUFFALO (Bit User Fast Friendly Aid to Logical Operations) Program communicates via the MC6850 Asynchronous Communications Interface Adapter (ACIA) Device and the MCU Serial Communications Interface (SCI).

The main module of the BUFFALO monitor program includes all the parts required by any of the individual command modules. Thus, the main module is a kernel of the BUFFALO Monitor program and consists of the five following parts: Initialization, Command Interpreter, I/O Routines, Utility Subroutines, and Command Table.

The MC68HC11A8 is manufactured using a unique combination of HCMOS and EEPROM technologies, offering the most cost effective system level solution available. On-chip memory systems include an 8K byte ROM, 512 bytes of EEPROM, and 256 bytes of static RAM. The inclusion of EEPROM allows field and factory calibrations to be stored on the chip.

Highly sophisticated, on-chip peripheral functions feature an eight-channel, 8-bit analog-to-digital converter, an enhanced set of serial ports, timer functions, and parallel ports with full hardware capacity. The MC68HC11A8 also provides an 8-bit Pulse Accumulator Circuit, as well as a Computer Operating Properly (COP) Watchdog System.

Providing highly sophisticated on-chip peripheral functions, low-power consumption, high noise immunity, and high-speed operation, the MC68HC11A8 has experienced enormous market acceptance since its introduction 4Q 84. Significant design-ins spanning all market segments are ramping into production. The ROMless version of the MC68HC11A8 is also available at even lower prices in production quantities.

MOTOROLA ARCHITECTS THE MC68HC811A2 MICROCOMPUTER

WITH ENHANCED USER EEPROM

Austin, Texas, May 12, 1986... Providing 2K bytes of user EEPROM, Motorola Microcomputer Operation's newest member of the M68HC11 Family, the MC68HC811A2 microcomputer (MCU), is designed specifically to facilitate user needs while retaining overall flexibility.

In addition to the large E^2 memory, which will allow field and factory calibrations to be stored on the chip, the MC68HC811A2 MCU also contains 256 bytes of RAM, an enhanced 16-bit timer system with three input captures and five output compare functions, and an 8-bit pulse accumulator. An enhanced asynchronous Serial Communications Interface (SCI), a synchronous Serial Peripheral Interface (SPI), an 8-channel 8-bit analog-to-digital (A/D) converter, parallel handshake ports, a bootloader ROM, and Real Time Interrupt (RTI) circuitry are also provided.

The MC68HC811A2 MCU utilizes an enhanced M6801 instruction set (88 new opcodes) which includes bit manipulation and 16-bit by 16-bit divide instructions. This MCU also features software programmable power saving modes, (WAIT and STOP).

The fully static design of the MC68HC811A2 allows operation of frequencies down to dc, further reducing its already low power consumption. It is designed to run at a 2.1 MHz bus speed across a -40°C to +125°C temperature range, to deliver powerful software control capacity in the harshest environment.

Available for sampling now, the MC68HC811A2 is initially priced in the $125 range for low quantities. Mr. Brian P. Wilkie, the Product Manager for the M68HC11 Family, states: "The MC68HC811A2 would experience the same steep learning curve as is being enjoyed by its ROM-Based relative the MC68HC11A8. So, prices should fall quickly."

M68EC11EVB

E^2 = MC

AUSTIN, TEXAS, MAY 12, 1986... Motorola's recently introduced M68HC11EVB is an evaluation board for the MC68HC11 single-chip MCU. The EVB allows the user to communicate and control the board from a terminal via a RS232-C interface, while helping them become aware of the benefits of EEPROM on MCU's.

The monitor features a line assembler/disassembler, a trace feature, multiple breakpoint setting, down load commands, memory and register displaying and modifying, and a user help command. By connecting the EVB to a target system, such as an IBM PC, the user may emulate the MC68HC11 in the single chip mode of operation. Together the assembler and the EVB provide a very economical means of writing, downloading and debugging user code, and evaluating target system performance. The board may also be used as a stand alone controller, much as in a distributed network processing system.

The EVB is available through authorized Motorola Distributors for $168.11. However, TWO THOUSAND M68HC11EVBs are now available at Distributors at a special sale price of $68.11 with coupon. Coupons are available through local Motorola Sales Offices, authorized Motorola Distributors, and current advertisements. For more information, contact your local Motorola Distributor.

Motorola's I/O Channel Modules Supported by MUSTANG-020™

The DATA-COMP MUSTANG-020 Super microcomputer has added another popular and important method of data acquisition, to its long list of accomplishments. Available within the very near future will be an adaptor module (from DATA-COMP) to interface with the super efficient I/O handlers known as Motorola Channel Modules.

Basically they provide the following features:

1. 12-bit address bus
2. 8-bit bidirectional bus
3. Asynchronous operations
4. Up to 2-megabyte transfer rate
5. Four interrupt lines
6. Reset line
7. 4-Mhz free running clock line

OBJECTIVES...

The I/O specifications define an interface system between the 'slave' devices and a primary system (MUSTANG-020, etc.). What this means to most of us is this: **we now have available a 'standardized' bus,** allowing practically unlimited expansion capability of the MUSTANG-020 (Euro-bus, 64 pin ribbon cable, etc).

The Motorola specifications define a communications path through which the 'core' system can communicate with "add on" I/O devices.

a. A 'core' system may perform hi-speed read and write operations with a slave device without disturbing any other slaves or its internal activities.

b. Specifies the electrical and mechanical constraints upon the design of master and slave units.

c. Specifies protocols that define the master slave relationship.

d. Provides terminology and definitions describing I/O channel operations.

THE BUS...

Typically the bus can be a 64 pin ribbon cable, with lengths up to 12 feet common. Various I/O devices are cable clamped (crimped) directly to the ribbon cable and inserted into retainer cabinets, head first. See figure 1-1.

A subsystem slave has two 50 pin ribbon cables on its interface system. It has its own power supply (as must longer runs of ribbon bus cable). It provides access to an internal terminator board allowing it to configure as the last slave on the system. See figure 1.2.

All data transfers depend on two interlocked signal lines - STB and XACK. STB is generated by the master and starts a data transfer. The XACK is generated by the addressed slave as an acknowledge signal. More detailed information may be secured from Motorola, Inc.

SOME TYPICAL Modules...

Serial & parallel I/O Module MVME400: Dual channel RS-232 communications module. Full duplex, sync/async baud rates of 50 to 19.2 KB. Configurable for terminal or modem operation.

Dual 16-bit parallel I/O Bus Module MVME410: Centronics compatible interface for printer applications. Four (4) 8-bit/2-handshake bit (40 lines).

SASI Bus I/O interface Module MVME420: SA400 type disk controller.

Magnetic Tape Adaptor Module MVME435A: Buffered 1/2" 9-track 4K bit FIFO buffer interface. Controls four (4) 25/125ips tape drives in start/stop mode.

A/D & D/A conversion modules: 12-bit conversion. 8/16 differential/single-ended channels. Four (4) full scale input ranges of 0.5, 1.5, or 10 volts. Accepts additional inputs of up to five (5) MVME601 Expander Input Modules.

A/D Input expansion module MVME601: designed to be used with the item above - MVME600.

12-bit D/A conversion module MVME605: four (4) channels of 12-bit D/A conversion. Five (5) voltage rangers, 0-0.5,0-10, ±2.5, ±10.0 volts. Two (2) current loop output ranges of 4 ma to 20 ma and 10 ma to 50 ma.

A/C input module MVME610: monitors the status of up to eight (8) 120/240 VAC sources. Max input is 300 VAC. Max isolation rating is 2500 VAC.

A/C output module MVME615: Zero crossover switching. Switches eight (8) independent outputs of 120/240 VAC. Max current switching is 3 Amp RMS.

MVME616: same as MVME616 but without zero crossover switching.

DC input I/O module MVME620: Eight (8) input channels for 10-60 VDC signal monitoring. 2500 volt input isolation. Input overvoltage and transient protection.

D/C output module MVME625: eight (8) 10-60 VDC output channels. 2500 volt input isolation. Inductive load transient suppression. Overcurrent protection of 2 amp max.

As you can see this expands the capabilities of the MUSTANG-020 vastly. As Motorola expands its I/O Channel Modules catalog, more devices will be available for the MUSTANG-020. This and other advanced features make the MUSTANG-020 a system that will be a world-leader, not only for price but super efficiency and utility, for years to come. **You can depend on DATA-COMP** to be out front in the "leading-edge" race.

Ordering information, **I/O Channel Module Bd.**
PRICE: $195.95
Direct from DATA-COMP

- - -

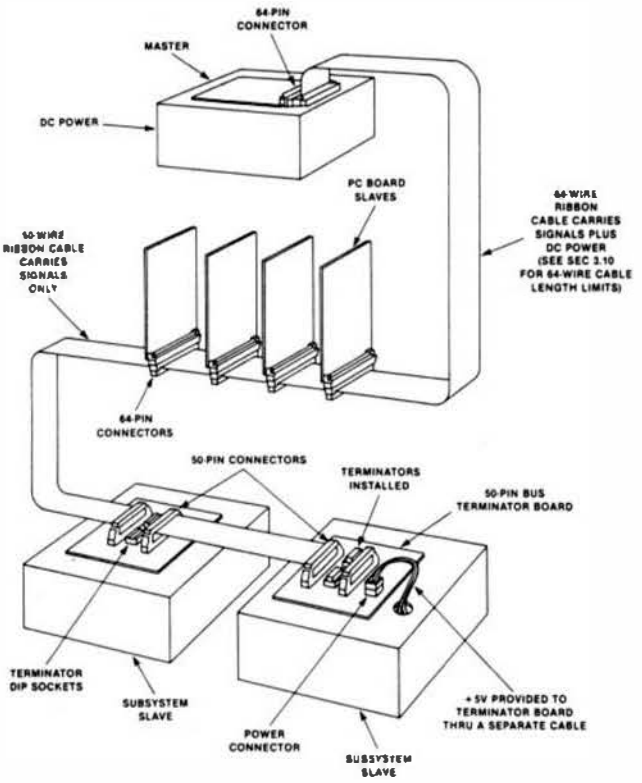


FIGURE 1-1. Typical I/O Channel Configuration

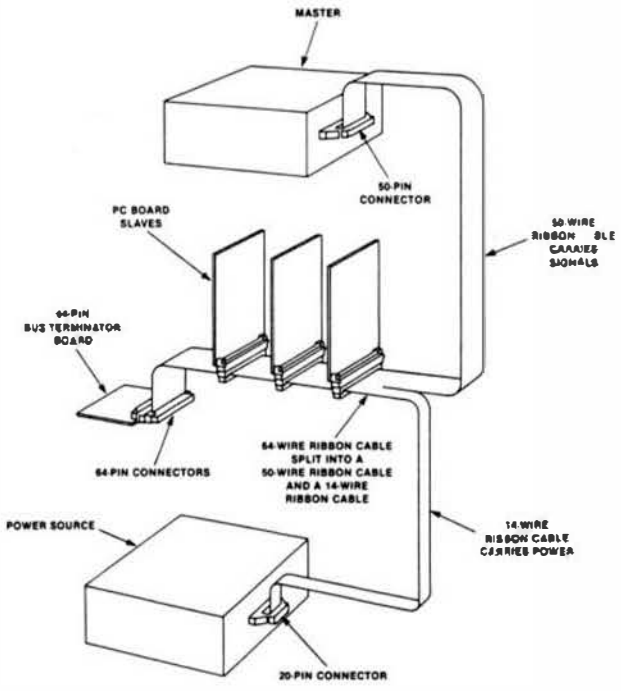


FIGURE 1-2. Alternate Power Wiring Configurations

microware
MICROWARE SYSTEMS CORPORATION

NEWS RELEASE

MICROWARE SYSTEMS CORPORATION
Ken Kaplen
515-224-1929

SUBJECT

MICROWARE PLAYS KEY ROLE IN NEW CD-ROM STANDARD

DES MOINES, IOWA -- Two electronics industry giants have announced a revolutionary new CD-ROM standard incorporating technology developed by Microware Systems Corporation. Philips N.V. of The Netherlands and Sony Corporation of Tokyo, Japan have unveiled a new type of compact disc capable of holding not only hi-fi sound but video pictures and computer data as well.

The new product, called "Compact Disc Interactive Media" ("CD-I"), heralds a new era in entertainment, education and electronic publishing. Based on the small, sturdy, ultra high fidelity compact disc which is currently sweeping the audio world, CD-I adds natural pictures and interactive capabilities. This will open the way for an important new method of distributing information. Some possibilities are talking encyclopedias, dictionaries, textbooks, electronic magazines and catalogs, as well as exciting new entertainment titles such as "music video" style presentations, songs with words and computer games with life-like realism. This new technology may also offer other possibilities as yet unimagined.

The interactive aspect of the media is especially significant as it allows the user to rapidly locate information and communicate with the highly intelligent CD-I player. The basic player is as easy to hook up and operate as a video cassette recorder, plus it can be optionally expanded to a full-feature personal computer. The new CD-I players are completely compatible with all current CD digital audio discs and the standardized format assures that any disc can be used with any player.

CD-I discs are also distinguished by their enormous storage capacity. Each inexpensive disc is capable of holding over 650 million characters of data. This translates to over 150,000 printed pages of text or up to 20 hours playing time of speech quality sound.

Philips and Sony originally co-developed the CD digital audio format which was first marketed in 1982. Because the new CD-I media relies heavily on advanced software technology, Philips and Sony invited Microware to participate in the CD-I development program. Microware contributed its special expertise in the creation of the control software for the microcomputer-based CD-I player and the information storage format for the discs.

CD-I players are based on the 68000 family microprocessor plus new specialized LSI components developed by Philips and Sony, including audio and video co-processors. The software is based on the kernel of Microware's OS-9/68000 operating system plus a newly developed CD-I file manager which is tailored to the special characteristics of the read-only media. For example, it allows any file to be opened with only one access even though the file system can have many levels of directories.

The Microware-developed system software package also fully supports the multiple audio and video formats including mixed data types within a single file. Video display capabilities include both natural pictures (delta YUV) and bit-plane graphics (RGB and color lookup table) in normal (384H by 280V) and high

resolution (768H by 560V) modes. Four digital audio quality levels range from ultra high-fidelity to high grade speech. This allows software designers to select audio and video quality levels which optimize storage requirements and playing time for the material to be presented.

It is expected that many major consumer electronics and personal computer manufacturers will obtain licenses to manufacture CD-I players. The basic player software will be included in the master licenses issued by Philips and Sony. Manufacturers may obtain licenses directly from Microware for the optional extensions which upgrade a basic player to a full-feature personal computer configuration.

OS-9 is a modular, real-time/multitasking operating system for computers based on the 68000 family of microprocessors. It is compact, ROMable and provides a Unix-style environment for application software. Since its introduction in 1983, OS-9/68000 has been licensed to over 200 OEMs for use in a wide variety of industrial, scientific and consumer products. Some of these products include personal computers, industrial control systems, data processing equipment and telecommunications systems.

Founded in 1977, Microware specializes in the development of advanced 68000 family operating systems and programming languages. Microware's offices are located in Des Moines, Iowa and Tokyo, Japan.

Messrs '68 Micro Journal
5900 Cassandra Smith Road
Hixson
Tennessee 37343
U.S.A

Dear Sir/s,

Let me take this opportunity to thank you for providing an invaluable service to the 68XX community, which must be unparalleled in the industry.

Due to the recent article on disk interleaving, I was able to speed up the disk read on two machines by a factor of 3! As a result the machines are much more pleasant to use, not to mention the time savings!

Kindly send me the following:

Flex user notes with 5" disk		$20.90
Reader service disk - 19		$12.50
Foreign Air S/H		$ 7.00
	US	$40.40

Enclosed is a draft for the above amount.

Yours sincerely,
MICRO CONTROLS LTD

LOTHAR THOLE
DEVELOPMENT ENGINEER

Dear Don,

Nothing at all to do with my series on XBASIC - just a follow-up to the recent discussions on the subject of re-executing the last-entered FLEX command (without re-loading the command itself). First, Jon Larimore's fix has a typo, in that the line 'ADDB 2' should read ADDB #2, otherwise the pointer is going to get bumped by the contents of memory-location 2, instead of by the number 2. However, there is a much neater and more compact routine which involves a change to only 9 bytes of 6809 FLEX to make a single key, LF (Line-Feed), do exactly the same thing . Make the following changes :-

	FROM	TO	
Address CE78	86 0D	BD CD27	Get next char
	BD CF41	2F F8	Repeat if alpha
	86 20	6E 9F CC1E	Jump to command
	20 C9		

A further enhancement, which will also tie in with the above mod, involves changing 3 more bytes in FLEX, and burning a small routine into your MONITOR's EPROM. This will add a ^R command, which will re-call the last entered command-line without executing it. The command-line will be recalled from the current cursor-position only so that if, for instance you accidentally entered

DOPY 1.FILENAME.TXT 2.FILENAME.ASM

instead of 'COPY', FLEX would come back with a 'BOT FOUND' message. You would then type 'C', followed by ^R, which would recall the command-line from the second letter onwards. A final entry of CR would cause the line to be executed. Now suppose you wanted to repeat the line, this time sending the file to Drive 3 instead of 2, you would enter ^R to recall the line, then cursor-left (or BACKSPACE) over the '2', type a '3' to overlay it, and then another ^R to recall the line once more from this cursor-position onwards. You now have a choice of entering LF (to execute the line without reloading the COPY command) or CR (to re-load COPY, and then execute). One important point to note is that with the LF option THE LENGTH OF THE COMMAND-LINE MUST NOT BE CHANGED, as you will not be entering another CR to indicate 'End-of-Command'. Anyway, here are the changes to FLEX, followed by the code to burn into EPROM (or locate in some out-of-the-way spot in memory) :-

	FROM	TO	
Address CE34	B1 CC01	7E xxxx	Jump to routine

BURN INTO EPROM

Address xxxx	B1 12		^R?
	27 0C		Yes. Go to R1
*	B1 02		No. Cursor-left?
	27 13		Yes. Goto R2
*	B1 05		No. Cursor-right?
	26 14		No. Go to R4
	30 01		Shift right in Line-Buffer
	20 0D		and branch to R3
R1	A6 80		Get char from Buffer
	B1 0D		CR?
	27 05		Yes. Go to R2
	BD CD18		No. Display this char
	20 F5		and back for next char
R2	30 1F		Shift left in Line-Buffer
* R3	7E CE31		Back to $CE31 in FLEX
R4	B1 CC01		DEL char?

```
*    10 27 DB5F Yes. Branch to $CE56 in FLEX
*    7E CE39    No. Go to $CE39 in FLEX

* Change to suit your Cursor-Control codes
* The HEX addresses are relative to $CE34. Change
  to suit your version of FLEX. The code DB5F in
  the last-but-one line assumes that the routine is
  burned in at starting-address $F2D0. Adjust this
  to suit your location!!
```

The spur which prodded me into developing the above routines was provided by an early attempt by Barry Guthrie (of Rhodes U., Grahamstown, Republic of South Africa) to implement recall and immediate execution of a command-line with ^R, but which, unfortunately, required that the command-line be correctly entered in the first place, as his routine prevented FLEX from seeing a BACKSPACE. Thus an entry of CAR (oops), BS, T would be seen by FLEX as CART, although the screen display would respond correctly and show CAT. Thank you, Barry, for stirring me into action!

Sincerely,

Bob

R. Jones
President

BAS-EDIT

All Flex users know that TSC BASICs are among the best interpreters available, and even compiler users find that it's frequently the quickest solution to some programming problems. We also know that it has an irritating quirk that begins with the fact that it only has a line editor and errors must be fixed by completely re-typing the line. Most get around this by using a separate editor to type in the lines of a program, after which another difficulty crops up -- BASIC (or XBASIC) will only load down to (but not including) the line with the error. So the game is: load BASIC, load program, find last line loaded, exit, load editor and program, fix error(s), then re-cycle. Well, this must have bothered John Roberts enough to do something about it and BAS-EDIT is the result.

BAS-EDIT comes with a well thought out AFFIX.CMD that is used to tailor BAS-EDIT to the terminal with which it will be used. (Note that I didn't say "install" or "modify" -- this is different). It includes a large variety of terminal (VDU) specific files and if yours is among these, it is only necessary to enter its name in response to a prompt. Otherwise AFFIX will ask for a set of terminal command sequences and then a set of control key inputs. AFFIX then renames the original BASIC.CMD to .OLD and creates a new one.

The new BASIC can be used exactly as the old one was -- in fact it has not been modified except for the cold start entry point. But it now has a small program at $5000 that checks the command line for the BAS-EDIT option. That is, "BASIC +E [,program name]. With this option, BAS-EDIT proceeds to install itself and then relocates to high memory. Thus, no memory is used unless the editor function is actually needed.

Operation from this point is simple. A program may be entered in the usual way until an error is reported. Striking the "edit" key (say "") causes the line just entered to be recalled and it may now be edited using right/left cursor controls, insert/delete characters, or overlayed with new data; exiting with a CR. Also, any previous line may be called by line number and edited or a line number itself may be changed (this duplicates the line to the second line number). An "express cursor" makes appending to the end of a line a snap.

BAS-EDIT is a nice addition to an already excellent BASIC and AFFIX is so cleverly done that it could very well be used as inspiration for anyone writing programs that must be installed on a variety of terminals.

Art Weller

NOTE: BAS-EDIT is available from S.E. MEDIA. See their advertising elsewhere in this issue. The price:

Limited Time!
SAVE $30.00
SPECIAL ONLY $39.95

STRIDE MICRO
680 South Rock Blvd.
Reno, NV 89520
Edward Chapin (702) 322-6868

STRIDE MICRO UNDER DIRECTION OF NEW CEO

RENO, Nev., April 28, 1986 - Stride Micro, the four year-old 68000-based supermicrocomputer systems manufacturer formerly known as Sage Computer, enters a new phase of corporate planning and development with the appointment of President and CEO, Edward O. Chapin.

Mr. Chapin assumes the responsibilities of Rod Coleman, former president and founder of Sage Computer who is now devoting full attention at Stride to the development of new products scheduled for introduction later this year. It was the technological expertise and insight of Mr. Coleman and two other Stride engineers, Bill Bonham and Bob Needham, that led to the 1982 introduction of the SAGE II - one of the first microcomputers based on Motorola's MC68000 microprocessor chip.

With a broad range of management, research and strategic planning experience in a variety of technical fields, Mr. Chapin brings to Stride the combined corporate/technology/marketing perspective needed to manage the company's growth.

As President and CEO of Kinder International, a technically-oriented retail photography company with outlets throughout the United States and Canada, Mr. Chapin built the operation into a $100 million a year company over four years. Prior to this, he was Associate Director with the University of California supervising a multi-disciplined scientific staff in research and development projects. In 1978, Mr. Chapin was appointed by the United States Secretary of Defense as chairman of the technology evaluation committee whose recommendations formed the basis for the President's "Strategic Defense Initiative (SDI) Program."

Heading up Stride's marketing and sales divisions under Mr. Chapin are Rhino Meyering, vice president of marketing and domestic sales and Rick Kriss, vice president of international operations. Mr. Meyering was previously vice president of marketing and sales at Ergo Systems Inc. and TTX Inc., director of sales at North Star Computer and western regional sales manager at Qume. Mr. Kriss served as vice president of finance and operations at OSAGA, Inc. and president of Penna, Inc. prior to joining Stride in 1982.

According to Mr. Chapin, Stride will maintain its current distribution network through VARs and OEMs, emphasizing the outstanding price/performance benefits afforded by the Stride series. He said the company's 1986 plans call for several new product introductions, a soon-to-be-announced modified pricing program and a renewed advertising/marketing support campaign.

Since its inception in January 1982, Stride Micro has developed, manufactured and delivered consistently superior high performance supermicrocomputer systems ranging from a single user system with up to 2 Mbytes of RAM to a 16-user system with 12 Mbytes of RAM and 440 Mbytes of disk capacity. Numerous operating systems including UNIX (tm) System V, p-System, RM/COS, CP/M-68K, Idris, PDOS, BOS, TRIPOS, FourBytePorth, Mirage and S1 are available for the Stride series. In addition, Stride Micro offers a Software Directory that encompasses many hundreds of vertical software packages covering applications such as accounting, medicine, banking, aerospace, design, manufacturing, astronomy, retail sales and artificial intelligence.

• • •

MEMBER
Association of Sleep Disorders Centers
wake
sleep
Organized 1975

SLEEP DISORDERS CENTER
STANFORD UNIVERSITY SCHOOL OF MEDICINE

William C. Dement, M.D., Ph.D., Director
(415) 497-6601

Dear Mr. Williams:

I was quite glad to see James Gross' article on the FFT in the April issue. There is a very confusing sea of literature on the topic, and Mr. Gross did an excellent job sorting it all out. I was especially pleased to see reference to the Uhrich algorithm, which I was not previously aware of: when I saw that the benchmark times for the algorithm were even faster than the Cooley-Tukey standard, I decided to translate Mr. Gross' code to C, and implement the Uhrich algorithm for a real time animal sleep scoring program I'm developing.

Unfortunately, Mr. Gross' program, as published, did not yield the same results as the standard DFT for the trial 12.5% pulse data. Comparing Mr. Gross' algorithm with the algorithm as published in 1969 by Uhrich, I discovered two errata:

1. The first line of the "i" loop (on page 45) has the sign for an inverse Fourier transform; it should read

```
W := -2*pi/N;
```

for the forward transform;

2. The formulae for calculating the "M" term are not correct; since the multiplication is complex, there should be four multiplications overall. The code should read:

```
M.re := x[l,C].re*cosw-
   x[l,C].im*sinw;
M.im := x[l,C].re*sinw+
   x[l,C].im*cosw;
```

A couple of other improvements which are easy to implement, and improve the performance of the algorithm are:

1. Moving the "W" assignment statement outside of all the loops, since it only needs to be calculated once;

2. If the algorithm is being implemented in C, using the bit shift operators (>> and <<) to perform the power of two calculations. The p2() function can be rewritten as the macro:

```
#define p2(x) ((x)?(2<<((x)-1)):1)
```

3. Taking advantage of previous calculations, by changing the following variable assignments:

```
replace D with in2j globally;
C := B + (N DIV P2(j));
E := A + (N DIV 2);
```

By defining another variable for (N DIV P2(j)), even more speed can be obtained. Running an efficient coding of the algorithm on a 16 MHz 68020 with a 68881 for floating point should yield some really impressive benchmarks.

Sincerely,

Russell N. Van Gelder

Dear Dan,

I have recently implemented the printer spooler on my FLEX system, and to my disappointment, I found that it was not reliable. From time to time my printer would repeat a character a number of times.
I traced it down to my printer, which obviously was level-sensitive to the data strobe from the system.
Now, if the timer interupts the spooling task just when the printer routine has set the data strobe low, it will remain low for about 10 mS, thus fooling my printer to think that characters are still coming in. The enclosed listing of my printer drivers shows how to fix this problem by disabling interupts during generation of the data strobe.
At the same time I also fixed the problem of having the printer hanging after pressing the system reset (as reported by Ron Anderson (?) in chapter 5, p. 14 of the FLEX USER NOTES). This is due to the way of using the IRQA flag in the 6821 PIA as printer ready.
When the printer has accepted a character, it outputs an $\overline{ACK}$ strobe which in turn sets the IRQA flag. When you now press the system reset, you will also reset the PIA by the hardware, clearing the IRQA flag and thereby indicating printer NOT READY, although it is ready.
I fixed this by letting the printer initialization routine output one null character, which will be ignored by the printer, but also acknowledged with an $\overline{ACK}$ strobe, thus setting the IRQA flag.

Yours faithfully

Alex Petersen
Tollosevej 56
DK-4550 Asnaes
Denmark

```
                    *
                    * DRIVER FOR A PARALLEL PRINTER USING
                    * THE PIA PORT A AT F200
                    * THIS ROUTINE DISABLES INTERUPTS DURING
                    * GENERATION OF THE DATA STROBE
                    *
                    * V.86.03.26     BY ALEX PETERSEN
                    *

           F200 PIA     EQU   $F200     PIA ADDRESS

                    *
                    * PRINTER INITIALIZATION
                    *
CCC8                      ORG   $CCC8
CCC8 86             PINIT LDA   #$3A      SELECT DDRA
CCCA 00 3A                BSR   PINIT2
CCCC 86 3E                LDA   #$FF      SELECT ALL LINES AS OUTPUT
CCCE B7 F200              STA   PIA       STORE IN DDRA
CCD1 8D 05                BSR   PINIT1    SET DATA STROBE HIGH
CCD3 7F F200              CLR   PIA       SEND ONE NULL CHAR TO PRINTER
CCD6 20 1E                BRA   STROBE    AND RETURN VIA POUT
CCD8 86 3E          PINIT1 LDA  #$3E      SELECT DRA AND SETUP
CCDA B7 F201        PINIT2 STA  PIA+1     FOR TRANSITION CHECK
                           RTS

                    *
                    * CHECK FOR PRINTER READY
                    *
CCDB                      ORG   $CCD0
CCDB 7D F201        PCHK  TST   PIA+1     SEE IF PRINTER IS READY
CCDE 39                   RTS

                    *
                    * PRINTER OUTPUT CHARACTER ROUTINE
                    *
CCE0                      ORG   $CCE0
CCE0 8D EE          POUTX BSR   PINIT1    SET DATA STROBE HIGH
CCE2 35 81                PULS  CC,PC     RESTORE CC AND RETURN
CCE4 8D F2          POUT  BSR   PCHK      TEST FOR PRINTER READY
CCE6 25 FC                BCS   POUT      LOOP UNTIL READY
CCE8 7D F200              TST   PIA       RESET IRQA FLAG
CCEB B7 F200              STA   PIA       PUT DATA IN ORA
CCEE 34 01          STROBE PSHS CC        SAVE ORIGINAL INTERRUPT MASK
CCF0 1A 10                ORCC  #$10      MAKE SURE TO DISABLE INTERUPTS
CCF2 86 36                LDA   #$36      SET DATA STROBE LOW
CCF4 8D 0C                BSR   PINIT2
CCF6 20 E8                BRA   POUTX

                    *
                          END

0 ERROR(S) DETECTED

SYMBOL TABLE

PCHK  CCDB  PIA   F200  PINIT CCC8  PINIT1 CCD8  PINIT2 CCDA
POUT  CCE4  POUTX CCE0  STROBE CCEE
```

Dear Mr Williams:

Thank you for the replacement of the Dec.85 issue you sent me. Finally I found an opportunity to contribute somehow to your Journal and specially to the owners of the Compacta UNIBOARD. I am reporting in this letter a 'Hardware design BUG' in the Uniboard and the correspondig fix to it.

Until today I have bought two of these boards with Flex and OS9, for the scientific instrumentation lab. that I manage at this university.
The OS9 operating system came installed to startup with the video-monitor and the keyboard as the standard output and input paths. I rather needed the Terminal (serial port) to be the initial I/O. So I decided to make the necessary changes to the INIT module, since then I use the serial port (6551). Here comes the 'annoying bug'. Whenever you try to 'list' or 'dir' files, the terminal get stuck and after a while it shows up with 'ERROR #246' (device not ready). This problem does not let you run programs that use both the terminal and disk drives at the same time.

I've written several times to the people of Compacta, as usual there are of no help, they've never answered a letter.
So, I decided to trace the 'bug' with a logic analyzer. The problem shows up in the moment that the 6809 is polling for the device that is requesting interrupt service. When the -DRQT from the DMA inserts a command for stretching the E clock, but precisely one cycle before the 6809 tries to read the SR (Status Register, $EF41) of the 6551, at that very moment the address lines are valid, but even the 6809 is in a wait state the 6551 receives a falling edge pulse, so clearing the interrupt bit of the SR. Reason: the 6551, ph2 (27) is connected to a free running 1 Mhz clock instead to the stretched E clock!!. Now when the wait state finishes and the 6809 is going to do its last cycle (reading the SR) it encounters with a SR showing that no interruptions was requested. The 6809 who is polling devices at the interrupt table ($B100), finds no IRQ request from any device. This produces the interruption of any programs that uses 6551-6844 and finally ending with the message ERROR #246.

Here is the FIX you should do: Cut the trace of ph2 between 6522(U19,25) and 6845(U28,23) and solder a WW cable from 74LS00(U20,11) to 6522(U19,25) or 6551(U12,27). This problem suggest that the 6522 is also requiring this modification if interrupt driven.

I hope this information will help and save time to most of the COMPACTA users. The most useful magazine I know is 68 Micro Journal, sincerely. Please include more articles on 68000/020 assembler.

Yours Faithfully,

Prof. Geza Kolzhaker,
Apdo. Correos # 393,
Merida 5101. V E N E Z U E L A.

Clearbrook Software Group Inc.

(604) 853-9118
U.S.: 446 Harrison Street, P.O. Box 8000-499, Sumas, WA 98295-8000

CLEARBROOK SOFTWARE GROUP
INFORMATION MANAGEMENT SYSTEM
TECHNICAL UPDATE

Some users of CSG IMS have reported problems when trying to compile programs with many (more than 201 labels. More labels are used in version 1.2 of the screen form program generator to allow backward movement through the fields.

To solve this problem, you will need to call the compiler (ImsC) directly from the OS9: prompt (or selection 9 of the CSG IMS executive) and pass some additional parameters. (You cannot pass extra parameters from the Ims executive menu selection 5.) The -t=N option of the compiler will allocate additional pages (256 bytes) of memory for the label table.

```
OS9:imsc custform -t=12
```

The above line will allocate 3K of memory for the label table. Another option (-s=N) will allocate more memory for the expression stack. We have not encountered a situation where the stack has been too small.

Version 1.3 of the compiler will have more initial memory allocated for the label table.

ALDUS

Aldus' PageMaker® Named "Best New Use of a Computer" by the Software Publishers Association

SAN FRANCISCO, California (April 18, 1986) — Aldus Corporation's PageMaker®, a desktop publishing program, was honored as "The Best New Use of a Computer" by the Software Publishers Association (SPA) at its Spring Symposium. The SPA is a trade association that represents every major segment of the microcomputer software industry.

"We are delighted to be recognized by our peers," said Paul Brainerd, president of Aldus Corporation. Brainerd, who coined the term "desktop publishing," added, "We are especially pleased that PageMaker was cited as the best new use of a computer. PageMaker and the advent of affordable graphics-oriented microcomputers and sophisticated laser technology gave birth to the desktop publishing revolution."

Available at a suggested retail price of $495, PageMaker (for the Macintosh) helps users create, edit, design and produce typeset-quality, camera-ready art for all types of publications. It has been used to create memos, reports, advertisements, brochures, newsletters and even tabloid-size newspapers. A version for the IBM-PC is now being developed.

PageMaker, which sparked the desktop publishing revolution, is widely recognized as the leading desktop publishing product in the U.S. as well as abroad. It has been translated into eight languages and is sold around the world through authorized distributors and dealers.

Aldus Corporation, headquartered in Seattle, Washington, was formed in 1984 to develop, market and support desktop publishing. The company is located at 411 First Avenue South, Seattle, WA 98104, telephone (206) 622-5500.

Classifieds

Winchester 10 Megabyte Drives
Two (2) 10 Megabyte Hard-Disk Winchester Drives. Working - were removed for upgrade to larger drives.

1 - RMS Model #509 $275.00
1 - Seagate Model #412 $275.00

(615) 842-4600 Tom 9-5 EST.

LSI 68008 CPU card, "C" Compiler and Digital Research CPM/68K $350.

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $595.

MICROKEY Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p. 51 DEC. 84 68' Micro Journal. $1800.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS. $745.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $359.00 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1290.

1-CDS1 20 Meg Hard Disk System with Controller $1000.

(615) 842-4600 M-F 9 AM to 5 PM EST

•••

68008 HARD DISK SYSTEM - COMPLETE

512K 68008 system, 10 megabyte hard disk, Xebec 1410A HD controller, 80 track double side, double density floppy. Complete with cabinet/power supply. Taken in on Mustang-020 trade-in. Version 1.2 OS-9, Basic09, Stylo, Mail Merge, Spelling Checker, Dynacalc - like new - original price $2,900.00 range (advertised) - SPECIAL - ONLY **$1750.00.**
615 842-4600 - Data Comp, ask for Don or Tom.

•••

ΩANTEΔ blank Data Systems 68 boards. Alen Gordon, M.D. 1435 W. 49th Place, Hialeah FL 33012 (305) 822-1100

•••

For Sale: Complete working 6809 system. 64KB memory, video board, ZVM121 monitor, keyboard, pair 96TPI DSDD floppies (1.3MB capacity each), EPROM programmer, serial & parallel ports, clock/calendar, heavy-duty switching power supply. Professionally built from Data Systems 68 boards, new components. With much software, documentation, unpopulated modem and 256KB memory cards. $500 CASH and CARRY. Bob Boyd, RR1 Box 575, Kennebunkport, Maine 04046. 207-967-5563.

•••

Gimix G68 Controller $400; Heath H19 ANSI terminal $400; SWTP Chassis and PS $100; MP09B CPU $75; MF68 Disk Cabinet and PS $60 MPR EPROM Burner $75; 2Mhz 64K Static Ram $75ea, DS68 CPU $60 DS68 64K DRam $60, DS68 DMA disk controller $100, Dual Acia w/Baud Rate Gen $100, Acorn 168K Prom Board $75, Acorn Rom/Ram Board $75 Thomas PIA Board $30, Misc SS50 and SS30 proto boards $25, disk drives $50 ea (813) 961-7320

•••

COMPACTA Unlboard, w/FLEX and OS-9, 1-40TK DSDD, 1-80TK DSDD, Monitor, Keyboard, Power Supply, Quality Software, Manuals. $450 obo. (619) 569-4064.

•••

SWTPC S+, 512K, DMF3 controller, 6-MPS-4 serial ports $3500, DAMF-2 8-inch disk system $1000; CDS-2 40 Meg with controller $2000; S/09, 384K, 5-MPS-2 serial, 1-MPL-2 parallel, DMF2 controller, DAMF-2, 8-inch disk system $2000; 8212 terminal $400; S/09 chassis with power supply $100; Chassis with 24V supply for 5-inch and Winchester $100; Uniflex software included: Basic, Pascal, C, Dynacalc, Stylo, Spell, Calculator, USE, Utilities 1&2, Diagnostics. Complete setup $7000 or make offer on items. call 801-224-5252-ext.128 ask for Vern.

•••

MICROBOX II single board computer kit (unassembled). Includes bare board firmware eprom and utility disk, construction and operating notes, firmware source on disk and small "C" compiler w/7220a graphic libraries. A $319.00 value for $225.00. Joe Condon, 8101 Alpine Drive, Des Moines, Iowa 50322. (515) 278-4581.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, GeneratinR a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful ProRrams. Programs sometimes use multiple Languages such as a short Assembly Language Routine for readinR a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP

A Division of Computer Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

Coming Soon!

68000 products running under FLEX™

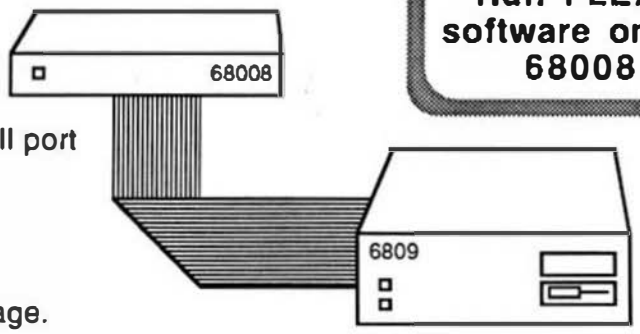
PLμS-68k (PL/9 for the 68000)

- Built-in screen editor
- Built-in source-level debugger
- Built-in assembler
- Byte, Integer, Long and Real variables
- Signed or unsigned variables
- Single-pass compiler
- Direct source to object
- Compiles over 1000 lines/min
- Requires second processor and any FLEX™ system with a PIA port

99% Compatible with PL/9

The second processor module (included with the compiler):

- 10MHz 68008 CPU
- 512K bytes RAM
- Case and power supply
- Plugs into Windrush UPROM-III port



Run FLEX™ software on the 68008

Interface software included:

- Program loader
- 68000 FLEX™ interface package.

Other Software:
68000 Assembler 68000 System Monitor
Programmer's Editor

For further information, phone or write:

Worstead Laboratories
North Walsham
Norfolk NR28 9SA
England

Tel (44) 692 404086
Telex 975548 WMICRO G

WINDRUSH
Micro Systems Ltd.

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.

Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Keyprgm, **Lifetime

Disk-3 Chug19, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Diskmove.

Disk-4 Mailing Program, *Findda1, *Change, *Testdisk.

DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.

Disk-6 **Purchase Order, Index (Disk file Indx)

Disk-7 Linking loader, Rload, Markmem

Disk-8 Crtest, Lampher (May 82)

Disk-9 Datecopy, Diskfix9 (Aug 82)

Disk-10 Home Accounting (July 82)

Disk-11 Disassembler (June 84)

Disk-12 Modem68 (revised June 86)

Disk-13 *Initm68, Testm68, *Cleanup, *Dskutlgn, Help, Date.Txt

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Compo)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc

Disk-17 Match Utility, RATBAS, A Basic Preprocessor

Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDDOUK, CMD.Txt (Sept. 85 Spray)

Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Dr, Log.Asm & Doc.

Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C

Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.

Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.

Disk-23 ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven Language Recognition Utility, Anderson March 1986.

Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.

Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.

Disk-26 Compacta UniBoard Review, Code & Diagrams. Burlingson March 1986.

NOTE:

This is a reader service ONLY! No warranty is offered or implied, they are as received by '68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

8" Disk $14.95 5" Disk $12.95

68' Micro Journal

5900 Cassandra Smith Rd. Hixson, TN 37343

(615)-842-4600

Telex 5106006630

*Indicates 6800

**Indicates BASIC SWTPC or TSC

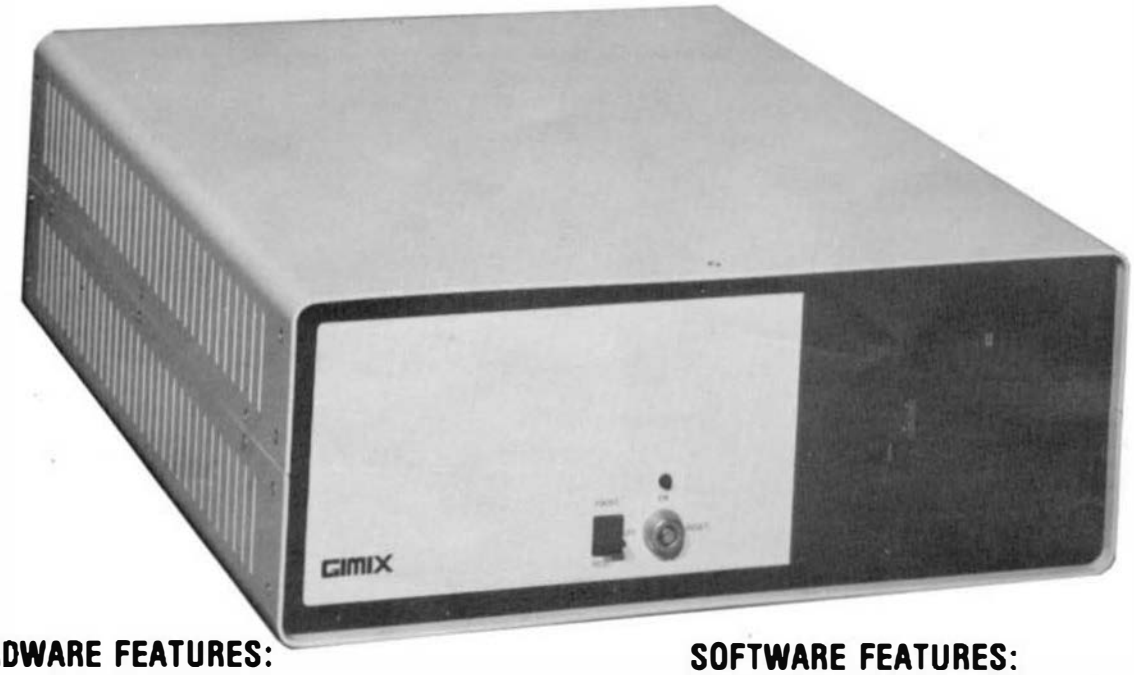
6809 no Indicator

Foreign Orders Add $4.50 for Surface Mail or $7.00 for Air Mail

*All Currency in U.S. Dollars

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

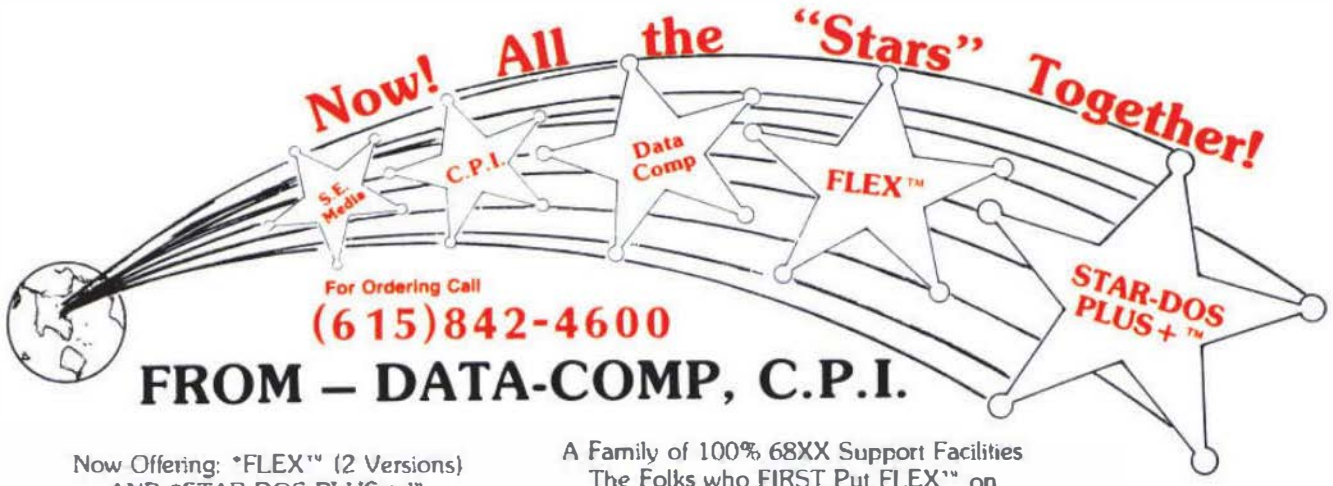
- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055



Now Offering: *FLEX™ (2 Versions) AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on The CoCo

FLEX-CoCo Sr. with TSC Editor, TSC Assembler, Complete with Manuals. Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr. without TSC Editor & Assembler **$49.95**

PLUS

ALL VERSIONS OF FLEX & STAR-DOS+ INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utlities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Sgstems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS. $469.95

* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS $129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K XXPAND TO 128K	$169.95
EPSON MX-MX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95

CABLES & OTHER INTERFACES CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

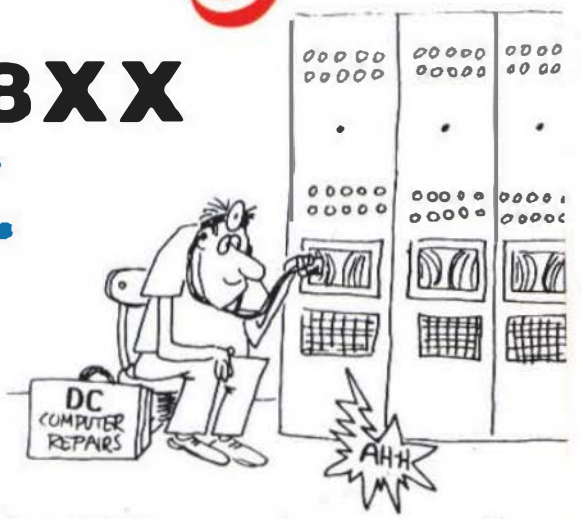
S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS



NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and confirm a **Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

This

Not This

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

☎ **(615)842-4607**
Telex 5106006630